CSG solid modelling and automatic NC machining of blend surfaces

Zhang, Dayong

Award date:
1986

Awarding institution:
University of Bath

CSG SOLID MODELLING AND AUTOMATIC NC MACHINING OF BLEND SURFACES

Submitted by

DAYONG ZHANG

For the Degree of PhD

of the University of Bath

1986

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

ACKNOWLEDGEMENTS

I am indebted and grateful to:

Dr J. R. Woodwark, who initialised and supervised this project for the first two years; it is he who introduced the author into the field of solid modelling. Without his guidance and influence, the author would not have become interested in this area.

Dr A. Bowyer, who supervised this project for the last year and guided and advised the author in many ways to fulfill this project. In particular, I would like to thank him for his patient reading and correcting of this thesis. Without his help, this thesis would never have been finished.

Mr A. F. Wallis, for his kind technical help and advice throughout the project. He also wrote the input language BLINP, which was used to create most of the models for this thesis.

Mr A. E. Carter and **Mr R. Pepler** for their kind help in the laboratory.

I would also like to thank numerous members of the department's staff and my research colleagues.

Finally, I would like to thank my wife for her understanding, encouragement and help during my research and the preparation of this thesis.

SUMMARY

A set-theoretic solid modelling system capable of representing three-dimensional objects consisting of simple and complicated surfaces and, equally importantly, blend surfaces, has been designed, developed and implemented. It has been tested graphically using raster scan display devices to generate shaded pictures, and practically on a three-dimensional NC milling machine to produce components with complicated and blended surfaces, turning the images in the designer's mind and on the screen into real objects.

The research work has included

a. the investigation of the geometric and algebraic characteristics of various solids defined by implicit polynomial inequalities of different orders,

b. the exploitation of mathematical techniques to define different kinds of solids, among which were solids with blended surfaces,

c. the design and testing of several data structures and algorithms for inputing, representing, storing and manipulating three-dimensional geometric information on various engineering components,

d. algorithms for producing various graphics including shaded pictures and

e. the development of methods that automatically generate NC machining instructions for controlling the process of manufacturing the engineering components defined using this system.

The blend surfaces generated by this system are continuous at least in both position and slope, so that they link surfaces together smoothly. The control of the

range, extent and bias of a blend can meet almost all requirements in mechanical engineering. The generation of NC machining instructions was entirely automatic and the linkage between the computer and the NC machine was direct. The NC machining instructions produced by this system were free of interference between the tool and the work. The precision and accuracy of the surfaces being machined could potentially meet any requirement, but is, in practice, limited by the precision and storage of the computer and the precision of the NC machine.

Research and experiments have shown that the techniques used in the solid modelling system described in this thesis are effective methods of handling three-dimensional geometric information about rigid engineering components. They are new and important contributions to CAE systems.

CONTENTS

CHAPTER 1

INTRODUCTION

Geometric modelling is a technique that uses computers to represent, store and manipulate geometric information about the shape of various objects. It is required in mechanical engineering in order to automate the manufacture of all kinds of engineering components.

Blend surfaces are non-ignorable parts of many engineering components. They are there to meet functional, manufacturing or aesthetic requirements, or their combinations. The representation and manipulation of such surfaces, and their integration with other kinds of surfaces are essential for the complete, accurate and reliable modelling of various engineering components. Successful achievement of this would be a valuable contribution to the automation of design and production systems.

1.1 BACKGROUND TO SHAPE MODELLING

Shape, or geometric form, is the basic (and probably the most important) feature of discrete, rigid objects, examples of which can be easily found in everyday life; for instance: a wine bottle, a wooden chair, a tin-opener or a car body. Physical shape modelling has long been used in various human activities. Small scale physical models made of clay, wax or wood have been used since pre-history to represent objects, such as architecture, ships, vehicles and all kinds of engineering components. Usually a large scale object is modelled by a smaller one; a hard object by a softer one; and an expensive one by a cheaper one.

The basic role of modelling has been to save time, effort and material in the design, construction or production of an object, but in practice physical modelling itself is by no means quick and cheap. For example, it may take a skillful person several days (or even weeks) to make a model. As human research and production activities become more and more sophisticated, products are designed and changed more frequently and produced more quickly. Furthermore new kinds of models are also increasingly needed for other purposes, such as molecular structures, astronomical phenomena, nuclear reactions and so on. To meet these requirements and to catch up with the rapid progresses on efficiency and accuracy of design-production technology, conventional physical modelling techniques are no longer completely satisfactory. New and more advanced modelling techniques are demanded.

As computer technology progresses, computer modelling techniques are improving and developing rapidly. They are playing more and more important roles in many areas. Computer-aided shape modelling [107], or simply *shape modelling*, is one of the major computer modelling techniques (examples of others are process modelling, engineering animation [65], fluid flow prediction, dynamic analysis, finite element analysis [37], and temperature and combustion modelling.). It has attracted the attention of many mathematicians, scientists and engineers in various fields from

theoretical research to industrial applications. Within a short period (about 1970 to the present day) it has brought about many fruitful achievements and is still an active, prominent and spreading research subject [77][78][4].

Geometric modelling techniques are particularly important for the design and production of various rigid products, specially in mechanical engineering, where the geometry of products needs to be precisely described, [96][59][12][42]. Algorithms for recognition [9], formulation, construction, organisation and manipulation of the geometric information about engineering components are needed to enable them to be handled automatically. The terms CAD (Computer-Aided Design), CAM (Computer-Aided Manufacture), and more generally, CAE (Computer-Aided Engineering) refer to the general application of computer technology in engineering production. Shape modelling has formed one of the most significant parts of the development of CAD/CAM/CAE systems. With the assistance of such systems one can do one's job much more efficiently, accurately and effectively.

As mentioned above, advances in geometric modelling are largely due to the development of modern computer technology. However, the outstanding difficulties that bother geometric modelling engineers are very often encountered in other areas rather than in computer science itself.

For example, as the geometry, or shape, of an object is the main source of the data that a modelling system has to deal with, mathematical representation has always been the central part of geometric modelling systems. One of the main tasks in developing a geometric modelling system is to explore, theoretically and practically, the mathematical techniques that are suitable for representing both simple and complicated geometric entities and their combinations.

It is well understood that an appropriate combination of the analytic power of mathematics and the numerical power of computers is fundamental to the building up of a sophisticated and practically useful geometric modelling system. The

mathematical tools employed in a geometric modelling system depend very much on the complexity of the objects to be represented, which can be two-dimensional curves, three-dimensioral surfaces or volumes, or even *four-dimensional surfaces*, such as temperature as a function of the three spatial variables.

The simplest geometric entities represented using a computer are probably those on traditional two-dimensional engineering drawings. Techniques and devices have been developed to produce complicated engineering drawings quickly. Systems capable of doing such jobs are called *draughting systems*. Some of the two-dimensional draughting systems developed in the U.K. are DOGS, GDS, SWIFT II, MLD, GENIE and MICRO-DESIGNER [32]. In them, little serious geometry or mathematical techniques are used; objects are mainly described by their coordinates and dimensions directly. two-dimensional draughting systems are usually menu driven and require user interaction. The idea of a draughting system is to automate and speed up the preparation of traditional engineering drawings using advanced equipment, nothing being conceptually changed.

As computer technology has developed, people have realised that computers are under-used just doing this kind of job; they can also be used to store, represent and produce more complicated three-dimensional objects directly without converting them into two-dimensional projections, though obviously more sophisticated mathematical and algorithmic techniques are required. The development of some new techniques in this field is the work described in this thesis.

Geometric modelling is a rather general term used for the computer modelling of the shape of an object. Some other terms are also often used to refer to more particular aspects of representations and operations in shape modelling [77][78][107]. They are introduced briefly here and some of them will be discussed in greater detail in later

chapters.

1.1.1 Surface Modeling

Surface modelling is one of the major techniques used in representing three-dimensional objects. It has played a significant role in geometric modelling. Surface modelling refers to the computer representation of the surface (or part of the surface) of an engineering component. A surface can be represented mathematically by one or more equations, numerically by a mesh, or even topologically by its boundaries such as its faces, edges and vertices.

The simplest surface is a plane. Planar surface models, such as those using triangles and polygons or their combinations, are also called *face models*. Planes in different positions and orientations can be put together to form shapes like cubes and polyhedrons, which, with appropriate combinatorial and assembly techniques, can be used to form rather complicated objects[106][108][109]. Relatively simple geometry can be used to represent and manipulate these objects.

Planes can also be used by appropriate programmes to approximate certain kinds of curved surfaces, such as cylinders and spheres. However, if the shape of an object involves more complicated surfaces, or a higher quality of representations is required, curved surfaces have to be used. This demands more powerful mathematical tools and techniques as well as more sophisticated algorithms.

To represent complicated objects like aircraft fuselages, car bodies or ship hulls, curved surfaces that are more complicated than cylinders, spheres or general quadrics are required. These are often referred to as *sculptured surfaces*. Apart from conventional polynomials, a number of other surface types have been used for this. Analyses on characteristics of different surfaces and methods to generate them have been discussed [5][31]. Among them parametric surfaces, especially *bi-parametric patches*, have formed an essential part of many current systems [35]. Bezier curves,

Coon's patches, B-splines, Bicubics, Steiner surface patches and Bertrand surfaces are all well known parametric entities used in geometric modelling systems.

1.1.2 Volume Modeling

Volume modelling is another, different, method for the computer representation of three-dimensional objects. As the term implies, instead of thinking of three-dimensional objects in terms of bivariate surfaces or univariate edges which happen to be embedded in three-dimensional space, this method treats a three-dimensional object as a collection of one or more simple volumes in space. CSG (*Constructive Solid Geometry*) is a typical volume modelling technique. It uses simple pre-built blocks, such as cubes, wedges, cylinders and cones, known as *primitives*, to construct more complicated ones. To do so, these primitives are scaled, modified, transformed and combined together to form the final object [6]. In practice, volume modelling is particularly suitable for modelling machined parts in mechanical engineering and buildings in architecture. Such an object can be constructed by a program using data on conventional engineering drawings [11], two-dimensional outlines [95], or from a text description [99][15].

Though surface modelling is suitable for representing sculptured surfaces, it can give rise to ambiguous representations when several surfaces are put together to form an object, for the interpretation of surfaces is not unique in all sorts of situations. This is because that the object's surfaces have two distinct sides: *solid* and *air*, but this is not a fundamental feature of all kinds of mathematical surfaces, particularly surfaces in the form of parametric patches. Volume modelling resolves this problem and is good at unambiguous representation, but the complexity of objects that it can handle is limited, because the number of different primitives is small and usually the shapes of primitives are geometrically simple. Basically, contrary to a surface modelling system, a volume modelling system can handle fairly complicated assemblies, but less complicated

geometries.

1.1.3 Solid Modeling

Solid modelling refers to integrated modelling techniques based on the geometry of objects being modelled. On the geometric side, it handles both surface and volume problems; in other words, it is able to deal with complicated geometries as well as complicated assemblies. Therefore, it is particularly suitable for representing solid objects, such as engineering components, in three dimensions. It can also represent or calculate other kinds of information about such objects [59], such as their colours, materials, volumes, centers of gravity, moments of inertia, surface areas, weights and costs, for example.

Since the beginning of solid modelling, considerable progress has been achieved. A number of solid modelling systems are commercially available (PADL ., MEDUSA, BOXER and ROMULUS, for example) [4][78][67], and more and better systems are being written all the time.

Solid modelling programmes completely represent the geometry of a given class of three-dimensional objects. As a subject, solid modelling includes both the mathematical theory governing such representations and techniques for implementing modellers on computers. An ideal solid modelling system would include the best features of both surface and volume modelling techniques, and the result should be much more robust and versatile than either technique on its own.

1.1.4 Set-Theoretic Operations

Most solid modelling systems use set-theoretic operators to link shape elements together to form a model. Set-theoretic operators are the Boolean operators: *Union*, *Intersection* and *Difference*. Because of its logicality, intuitiveness and conciseness, set-theory, together with certain transformation (translation, rotation and scaling) and sweeping techniques (moving a geometric entity along a space curve), has

proved to be one of the most powerful and widely accepted input tools for many of today's solid modelling systems [Chapter 4]. It is also a convenient and effective method for representing solid models inside a modelling programme, though few systems have used it in this way completely, due to its lack of locality and some other weak points [Chapter 2].

1.2 CONSIDERATIONS IN DESIGNING A SOLID MODELLING SYSTEM

Different solid modelling systems are designed for different purposes. As far as mechanical engineering is concerned, solid modelling is considered one of the most important techniques in CAD/CAM/CAE systems. The characteristics of a solid modelling system should be judged by its performance in its applications. Therefore, the range of applications, quality of results and efficiency of performance should all be considered in designing a solid modelling system.

The requirements for a solid modelling system vary from one to another, but a good solid modelling system in mechanical engineering should at least be able automatically to calculate any well-defined geometrical property of any solid that it can represent, such as the mass, the volume, surface area or moment of inertia. It should also be capable of generating sufficient data for subsequent applications, for example, picture generation, functional analysis or process planning.

The geometric representation of engineering components is not the final goal of a system, but the basic requirement for all subsequent applications. Simplicity, precision, speed, compatibility and flexibility of a solid modelling system are all important for the design, analysis, communication, manufacture and documentation of engineering components.

In this thesis, in addition to requirements for various graphics, detailed considerations are given to one of the most important applications: the automatic generation of NC machining instructions. In mechanical engineering, especially in manufacturing industry, the automatic generation of NC machining instructions is one of the most desired and practical requirements. The idea using a solid modelling system to generation NC machining instructions is to simplify and to automate the process from design to production of engineering components, as well as to improve the complexity of the shape of components and the quality of products. To do this, the

information attached to a solid model must be complete, accurate and appropriate. The solid modelling system must be able to represent all kinds of surfaces that would appear on the final products and other information needed in the generation of NC machining instructions.

In mechanical engineering, most components have simple surfaces, such as planes, quadrics and tori. However, there are a great number of components which are not only just made of such simple surfaces, but also have blends or fillets, which are more complicated, between them. Castings, forgings and injection mouldings are typical examples of such engineering components.

To represent the large number of different kinds of components in mechanical engineering, a solid modelling system must be able to support both simple and complicated surfaces.

Volume modelling with primitive solids is suitable for representing machined parts, since the surfaces of primitives are geometrically simple and can be specified mathematically by low order expressions, such as planes and quadrics. But a volume modelling system of this type cannot handle complicated surfaces, such as fillets, properly.

There is widespread use of parametric surfaces as sculptured surfaces in the shipbuilding, aircraft and automobile industries. These might be useful for representing curved surfaces in a *set-theoretic modeller*. Despite this, parametric surfaces are inconvenient when the need arises to represent a low order surface such as those on machined parts, because parametric surfaces (B-splines, for instance) which are suitable for sculptured surfaces are difficult to use to represent low order surfaces (such as a cylinder) precisely and conveniently.

It is also difficult to integrate sculptured parametric patches with solid primitives in one system, because solid primitives are usually constructed using bounded implicit

polynomials, whereas parametric patches are defined by meshes in parametric space. This lack of definition consistency may cause serious problems in designing and implementing a system. For example, set-theoretic operations are used in constructing and manipulating a model. These operations use the property of an implicit function that it has an *inside* and an *outside* (distinguished by their potential values), but the inside/outside property of a parametric patch is neither obvious nor globally available, because a parametric patch is usually defined by a number of control points, or a mesh, which are localised over a topologically rectangular area. Therefore, it is difficult to make these two kinds of surface compatible.

Another consideration is the representation of blend or fillet surfaces. A blend surface, unlike a sculptured surface of a car body or ship hull, links other surfaces together smoothly. That is to say, it must at least meet the requirement of continuity in both position and slope. This is hard to achieve using parametric patches, because the continuity relationship between solid primitives and parametric patches is difficult to establish.

Techniques for smoothing free-form surfaces have been proposed to construct a smoothed surface or object from pre-defined patches or polyhedra by modifying their control points or other parameters [54][55] [8][86][34][92]. The smoothed patches can be connected together to form certain kinds of blend surfaces for blending simple objects, however this is not a convenient way for specifying blend surfaces between other surfaces where the blend is required to be controlled by the blended surfaces. It is equally hard to specify a blend parametric surface patch between another two parametric patches. When more than two surfaces have to be blended (such as at a corner where three surfaces meet) it is almost impossible to use parametric patches.

Conversions between certain classes of implicit polynomials and parametric equations are possible[85]. They are known as *parameterization* and *implicitization*. These conversions may improve the efficiency of some calculations, such as that needed

to find the intersection between two surfaces. However, because of the inconsistency of the inside/outside property of the implicit and parametric surfaces which are to be used for set-theoretic operations, this conversion may cause some ambiguities. For example, the value of a parametric equation can have positive or negative values for different points at the same side of the surface. It is difficult to decide if this side should be treated as *solid* or *air* when used with Boolean operations and also difficult to decide if this side should be negative or positive when converted to an implicit equation.

Since the relationship between blended surfaces and the blending surface is rather important, it is much more convenient if these two are the same kind of functions at the definition stage. This will be even more important for defining blend surfaces on other blend surfaces.

Problems considered in the above discussion have not been solved properly and completely so far, and new and more powerful techniques are demanded to develop and implement better systems.

1.3 PROBLEMS TO BE SOLVED IN THIS THESIS

From what has just been discussed, we can see some of the difficulties in developing a perfect solid modelling system. It needs a great deal of intellectual and experimental work even to develop part of such a system.

In this thesis, attempts are made to solve two main problems: The representation of complicated engineering components and the automatic manufacturing of such components. To do this, several requirements are distinguished and considered. They are individual and also integrated problems that require investigation both independently and relatively. To develop a solid modelling system which meets these requirements, the following targets were set up to be achieved in this work:

a. To establish an appropriate representation scheme and a data structure which are good at accommodating both simple and complicated surfaces and representing relationships between them, and which can be easily used in subsequent applications.

b. To develop techniques to describe and integrate both simple and complicated surfaces in one system compatibly. This is necessary for representing complicated engineering components that consist of both kinds of surfaces.

c. To develop techniques of defining and representing blend surfaces, whereby the blend surfaces are well attached to and controlled by the surfaces being blended.

d. To design algorithms to generate various graphics, including shaded pictures and section views.

e. To design algorithms automatically to generate NC machining instructions based on the solid model. This requires composite surfaces that consist of different kinds of surfaces to be machined without being split into individual pieces. This will eliminate the necessity of user intervention during the course of machining instruction generation.

These targets are dealt with in the following Chapters. A pure CSG representation scheme is used as the framework to hold the geometric information on the objects throughout all the stages of design, construction and representation of the model. This is discussed in Chapter 2. A multistage definition based on and developed from Liming's [60] method is used to solve the problems of representing different kinds of surfaces (simple, complicated and *corner blends*), and a new method is also proposed to represent *gap blends*. These methods and their ramifications are discussed in Chapter 3. Input techniques and methods are introduced in Chapter 4; these make the definition and construction of models more intuitive and convenient. In Chapter 5, some important algorithmic techniques are introduced, which greatly improve the efficiency of model creation and manipulation. Techniques of generating different kinds of pictures are discussed in Chapter 7. Data structures and algorithms for automatically generating NC machining instructions for producing modelled components are developed in Chapter 8. Finally, in Chapter 9, conclusions on the solid modelling system presented in this thesis are made and some suggestions for further work are put forward.

CHAPTER 2

REPRESENTATION SCHEMES

One important feature of a solid modelling system that distinguishes it from a draughting system is that it has an internal representation, or *model*, of the object. This model is a geometrically complete three-dimensional representation of the object and it determines the behaviour of the system in terms of geometric domain, efficiency and range of applications.

There are several ways to generate and hold a model. For example, an object can be viewed as a cluster of points in space, a combination of certain primitives, or regions bounded by surfaces. Different usage of the geometry of the object leads to different representation schemes. Among several different representation data structures [4] [20][50][77][78][101], are two kinds of representations which have been recognised as the main schemes used in existing solid modelling systems and are anticipated to remain so for some time in further generations of solid modelling systems. They are B-rep (Boundary Representation) and CSG (Constructive Solid Geometry) (Fig.2.1).

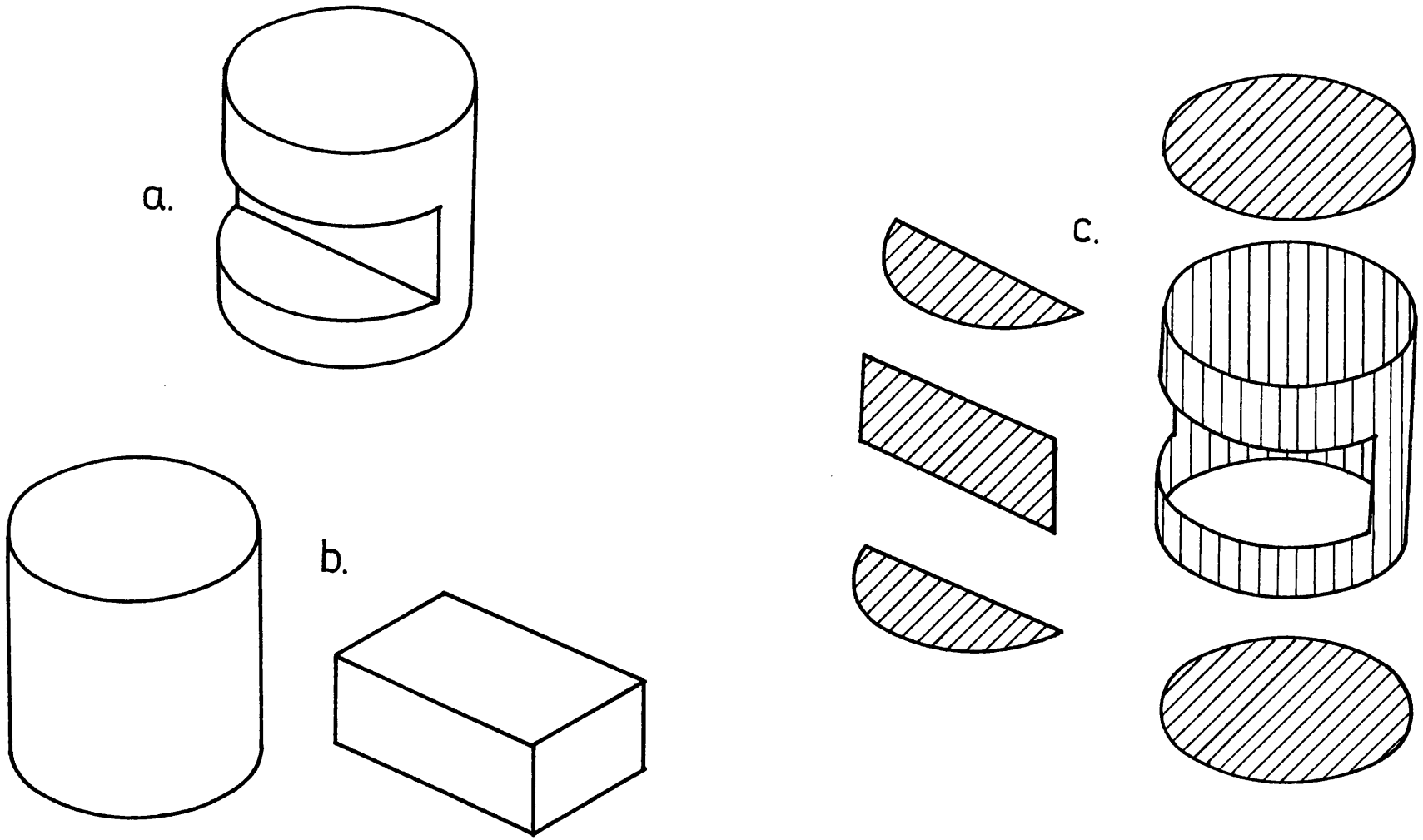


Fig.2.1 a an object b. elements for CSG c. elements for B-rep

2.1 BOUNDARY REPRESENTATION (B-REP)

A Boundary Representation assumes that the object is represented by a region in space bounded by surfaces, which, in turn, are considered areas bounded by edges and vertices [17]. By holding all vertices and edges, and hence all the faces of the object, the bounded region, which effectively represents the body of the object, can be worked out using proper techniques.

Most of today's solid modeling systems are of B-rep type and some of them are listed in table 2.1 [4][78][67].

B-reps are in fact collections of faces described by surface equations. These faces are held in a data structure which acts as the framework for holding the geometric information. This data structure topologically ensures the validity of the representation of the object.

To represent an object with a B-rep scheme, its surfaces must be related to each other properly to enable their edges and vertices to be calculated and stored.

If the input data are primitives or surfaces defined by implicit polynomials, relationships between surfaces can be defined by their set-theoretic expressions. Using appropriate algorithms, their intersections can be calculated and stored in a B-rep data structure. This approach is in fact a conversion from a CSG specification to a B-rep model.

If parametric patches are used, or the vertices and edges of a model are input directly by the user at a terminal (some of these data are available from conventional engineering drawings), apart from necessary calculations to obtain intersections between patches, topological operations are necessary to ensure the solidity of the model. The verified vertices and edges are then related to each other and stored in a B-rep data base.

Modeller	Vendor	Core Software	Genre
Catia	IBM	Dassault	B-rep
CATSOFT	CATRONIX		CSG
DDM-Solids	Calma		B-rep
DORA		University of Bath	CSG
Euclid	Matra/DEC	CNRS	B-rep
Geomod	SDRC/GE CAE	SDRC	
ICM GMS	ICM		B-rep
MEDUSA	Prime	CIS/CV(Cambridge)	B-rep
Modelmaker	Cubicomp		B-rep
Microsolid	Perspective Design		B-rep
Noname	Pafec	Leeds University	B-rep
Omnisolids	MCS		B-rep
PADL-2		Rochester University,NY	CSG
Patran-G	PDA Engineering		cell
Romulus	Evans & Sutherland	Shape Data, UK	B-rep
Solidesign	ComputerVision		B-rep
Synthavision	MAGI		CSG
Tips	CAMI	Hokkaido University	CSG
UNIS-CAD	Sperry Univac	Technical University, Berlin	B-rep
VOLE	Radan Compututational	University of Bath	CSG

Table 2.1 Some Current Commercial Solid Modelling Systems

Edges and vertices stored in a B-rep data base are more often handled in subsequent manipulations than the surfaces or patches themselves.

Vertices, edges and faces (a bounded part of a surface) are the three types of data that need to be handled topologically in a B-rep scheme. There are nine possible adjacency relationships between them. These adjacency relationships can be used in organising different kinds of data structures to hold the shape elements and to relate them to each other [97][51].

The *Winged Edge* structure (The name Winged Edge came from its graphic appearance, with the reference edge in the middle and adjacent edges and faces on both sides resembling the wings of a bird.), as shown in Fig.2.2, is one of the typical data structures used in B-rep solid modeling systems. It is particularly suitable for representing planar-faced polyhedral solid models. It can also represent certain kinds of objects with curved surfaces. Each object in the structure has pointers to adjacent objects, so the faces have pointers to their vertices, and so on.

There are other data structure types for holding these shape elements [97] [102], such as *vertex-edge* structures and *face-edge* structures, but the principle is the same. The basic requirements are sufficiency of the information and the efficiency of the modeller's operation.

Euler operations have been used for topological calculations in geometric modelling [61]. They are used in B-rep solid modeling systems to enable the data structure, based on the relationships between vertices, edges and faces, to be analysed topologically. This is very useful for ensuring the validity of the data representing the model.

The Euler equation for closed polygonal objects with holes in takes the following form

$$V - E + 2F - L - 2(Sc - H) = 0$$

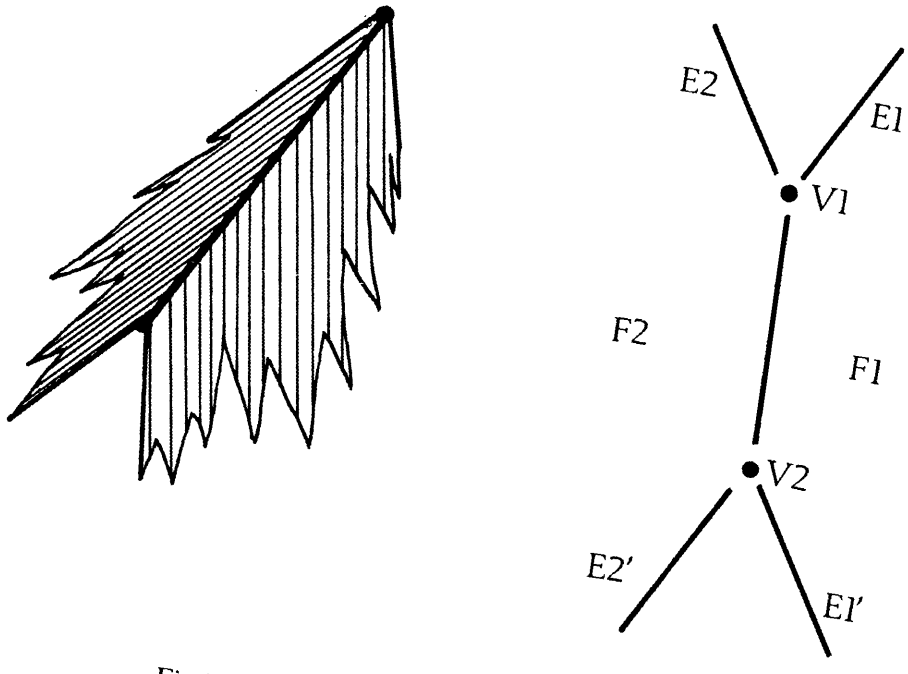


Fig.2.2 The Winged Edge Data Structure

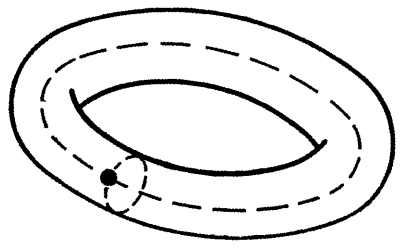


Fig.2.3 Dummy Vertex and Edges on the Torus

where V = vertices, E = edges, F = faces, L = loops, H = holes and Sc = number of closed objects [100].

If curved surfaces are included, the Euler equation becomes

$$V - E + 2F - 2(S - G) = 0$$

where S = number of shells (single connected, closed volumes) and G = the sum of the genus (number of holes) in the shells [100].

There are other different forms of Euler equations [89]. The particular formula used in a solid modelling system depends on the types of models to be represented, and the geometric tools and the internal representation used for the topological entities.

One very important requirements is consistency between geometry and topology. For example, a surface with only a single vertex as its boundary (a cone, for instance) has to be allowed if such surfaces appear on the object, or assumptions have to be made to allow the same Euler equation to be used in wider applications, such as the assumption of a dummy vertex on a sphere or edges and vertex on a torus where geometrically there is none (Fig.2.3).

It is not easy to generalise the Euler equations into a single form that could be used in any solid modelling system to model any kind of solid models for different applications. To allow more complicated geometric entities as well as different kinds of topological combinations to be used in a solid modelling system, it is necessary from time to time to modify the data structure, Euler equations, or model construction rules. The use of Euler equations in B-rep modelling systems does provide a stable method for constructing valid solid models, but, because of the particular form of Euler equation applied in a system and special rules imposed in model construction, it more or less restricts the range of objects that can be modeled.

The idea behind B-rep is to simplify the representation of a three-dimensional

object from the handling of spatial volumes to the handling of faces, then further to simplify handling their bounding edges and vertices. This simplification improves the behaviour of interactive manipulation and the speed of picture generation (mainly line drawings). It also has a relatively close relationship with conventional two-dimensional drawings. (In fact many B-rep systems have grown from considerations of engineering drawings.) This enables vertices and edges to be input directly, and makes it possible to convert simple two-dimensional engineering drawings to three-dimensional models.

B-rep techniques are relatively well developed. Many commercially available solid modeling systems use B-rep as their internal representation scheme. Since vertices and edges are explicitly held in the computer, B-rep modelers are usually fast at drawing wire-frame pictures, which are convenient for interactive design, analysis and robotics operation and monitoring. They are also quite fast for shaded drawing production, as, for example, a painter's algorithm can be used easily with them to create such pictures[64].

However, despite its widespread use, B-rep has some weak points:

a. The heavy load on programming and calculation of the topological operations for checking the consistency of vertices, edges and faces.

b. The difficulty of precise specification of high order surfaces. Vertices and edges are not the essential geometric features of many curved surface (the sphere and the torus, for example), but they are necessary for producing the boundaries of the model. The edges obtained by calculating the intersections of two curved surfaces, especially high order surfaces, are much more difficult and less accurate than those of planar ones. This is because, though edges and faces of a curve-faced object are topologically no different from those of a planar-faced polyhedron, they are geometrically more difficult to handle and may bring about numerical problems during calculation.

c. Redundant data maintenance on shape elements. Most B-rep systems store both the surface equations and the face boundaries. Also, more than the minimum necessary links between vertices, edges and faces are very often maintained to improve the efficiency of topological searches and checking. One modification in one place requires a simultaneous change in the others. This is both redundant and error prone.

d. Surface types are limited. Topology and geometry are not consistent in all places. To keep the consistency between them in a solid modelling system, and for ease programming, reliable topological operation, and numerical problem avoidance, surfaces are often restricted to certain types and their combinations have to follow certain restricted construction rules.

e. Difficulties in supporting blend and fillet surfaces. Blend surfaces link other surfaces together smoothly. Usually, they are neither symmetrical like a cylinder or the torus, nor are they singular (They often have more than one separate surface, eg. hyperboloid.). It is rather difficult to find the boundaries of blend surface using current techniques. Also, their use would *smooth away* some vertices or edges (Fig.3.23), or even part of an edge. This may bring problems with topological operations.

It is not an easy job to solve these problems in the further development of B-rep approaches towards much more sophisticated, versatile and effective solid modeling systems. Different approaches are worthy of being explored and investigated in order to achieve this goal. The other main modelling representation scheme, the CSG approach, is discussed in the following section.

2.2 CONSTRUCTIVE SOLID GEOMETRY (CSG) REPRESENTATION

CSG is a technique that constructs a complicated model from relatively simple blocks or primitives, such as the cube, cylinder, cone and the torus (Fig.2.4). They are built up beforehand and subsequently used as shape elements in the construction of the final model (Fig.2.5). (Simple B-rep models can also be converted to CSG models using appropriate programs [72].) set-theoretic operations are used to organise these primitives together to form the required object. Because of this, a CSG modeller is also called a *Set-Theoretic Modeler* [104].

A CSG modeller views an object differently, but more naturally, than a B-rep modeller in the sense that: firstly, unlike the B-rep approach (which views a model as collections of vertices, edges and faces in a topologically flexible *rubber universe*), CSG considers the model as a solid region strictly defined by geometry in a *rigid universe*, which is closer to the reality of our world; secondly, unlike the B-rep approach (which uses lower dimensional topological items, such as vertices and edges, in the model's construction), CSG uses volumes directly, to which surfaces are attached consistently without ambiguity; thirdly, no topological operations are needed and theoretically there is no difference between the construction of simple polyhedron and a complicated one with curved surfaces and an arbitrary number of holes in. Because of these fundamental differences, the CSG approach is more robust and reliable in CAD/CAM/CAE applications.

In designing a product with internal void regions, holes or ducts (such as a tap or a valve), the CSG scheme provides a means whereby the user can view the internal surface by sectioning the model with one or more planes at different position and angles (Fig.2.6a). The internal region can also be converted into a solid model easily using CSG techniques. Though most B-rep modellers allow this, too, CSG modellers find it simpler and easier. They are more intuitive and can accommodate much more complicated geometric entities. In practice, the user can construct the internal part first as a solid

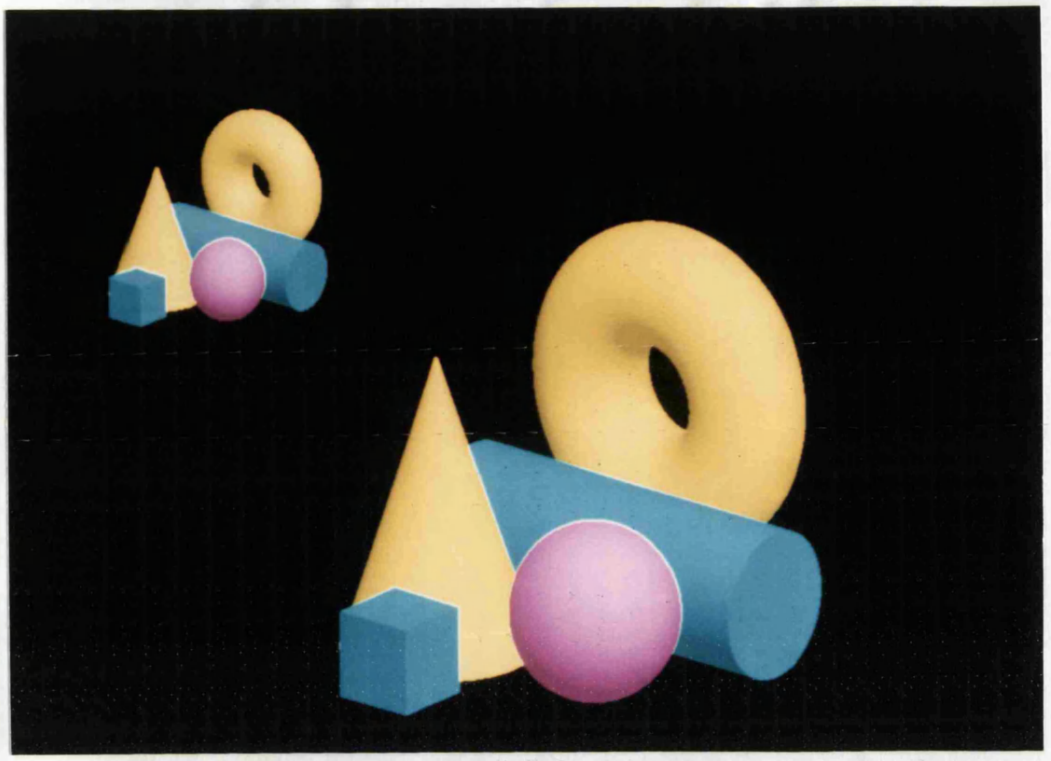


Fig.2.4 Some Primitives Used in a CSG Modeller

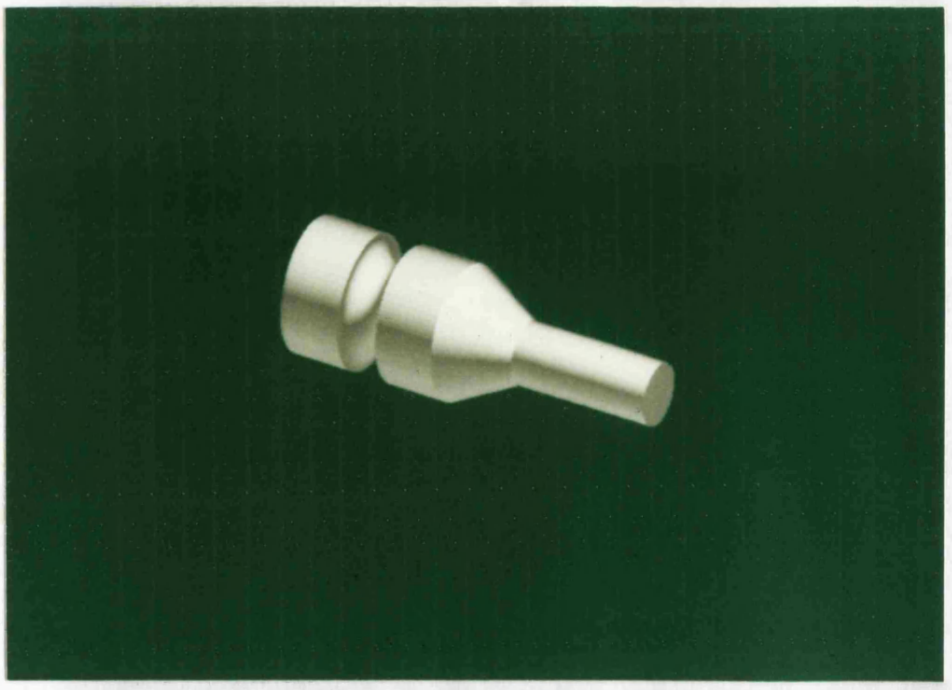
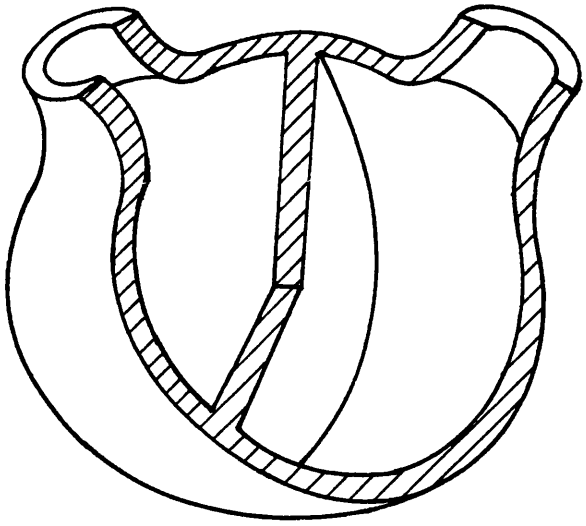
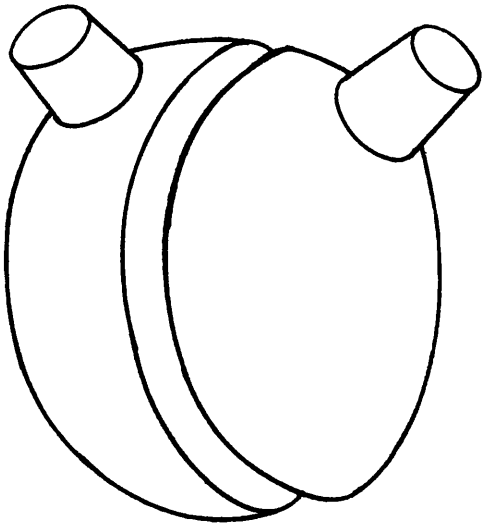


Fig.2.5 A Model Made of Simple Primitives



(a)



(b)

fig. 2.6 a section view and solids converted from voids

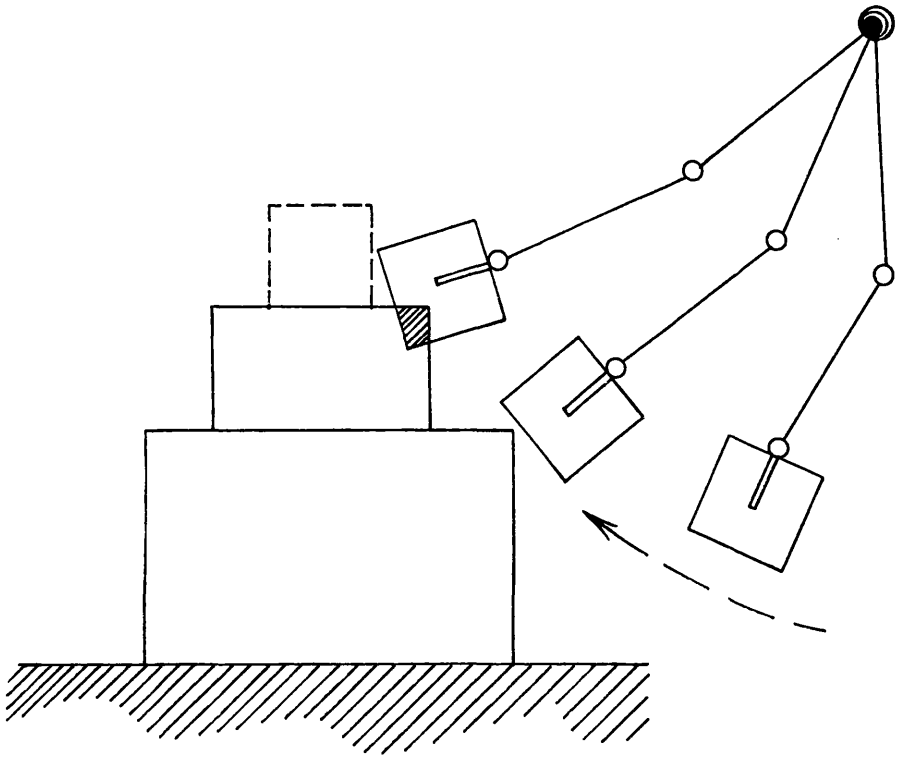
volume and then subtract it from the body of the model (Fig.2.6).

In manufacturing industry, material removal is one of the basic functions of all production processes on machine tools. Using the CSG technique, the shape and volume of the material to be removed can be extracted from the raw material by differencing from it the final model. This is useful to decide the shape and volume of the raw material block, the cutting sequence and feed rate, and also to plan the tool path.

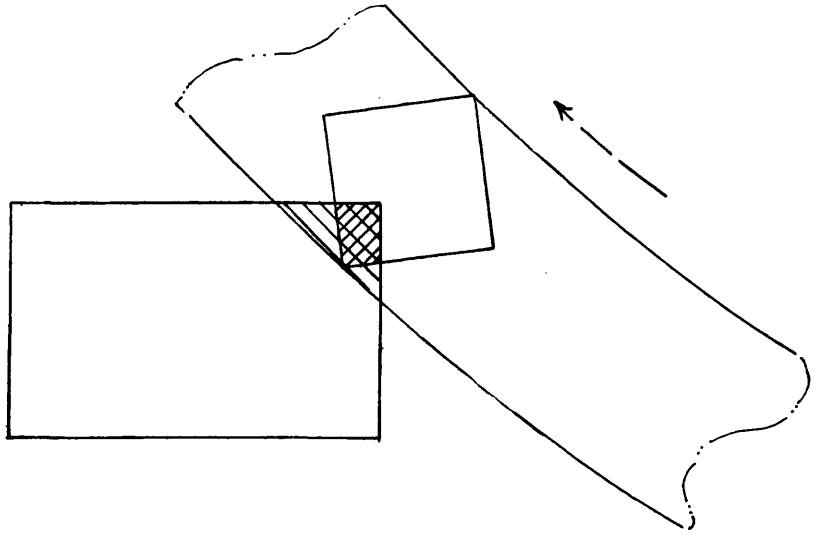
Another application of the CSG technique is the modelling of dynamic motion[114]. For example, in robotic movement analysis, interference detection is necessary. By constructing a model of the region swept by the movement of the robot, we can obtain its *motion field*. Checking the resultant volume of the set intersection operation on the static models and the swept volume of the moving part, we can find out if they interfere with each other and, if they do, where this happens and which the responsible parts are (Fig.2.7). This can also be applied to machining environment to detect if the cutter cuts the final object by comparing the volume of the object with the model of the *motion field* of the cutter.

The usefulness and convenience of the CSG technique can be easily seen in such applications. However, so far, CSG has mainly been used for the building up of models from primitives that are then converted to a boundary representation. As a technique in CAE systems, it has been less well explored and investigated than the B-rep approach. With appropriate efforts in mathematical analysis and algorithmic design and implementation, more sophisticated techniques can be developed based on the CSG theory to meet more difficult requirements in all sorts of applications.

Some solid modelling systems are perceived as CSG systems [table 2.1], but in fact most of them still use B-rep in at least part of their internal representation. Very few (e.g. SYNTHAVISION, DORA) use only CSG as their internal representation scheme. In contrast with the B-rep scheme, we assume that the CSG being discussed for the rest of this thesis is both an input tool and the only internal representation scheme



(a)



(b)

Fig.2.7 Robotic Interference Checking

in the solid modeller and that no B-rep is used at all.

The reasons that CSG has been widely used as an input technique but not as a popular internal representational scheme are as follows:

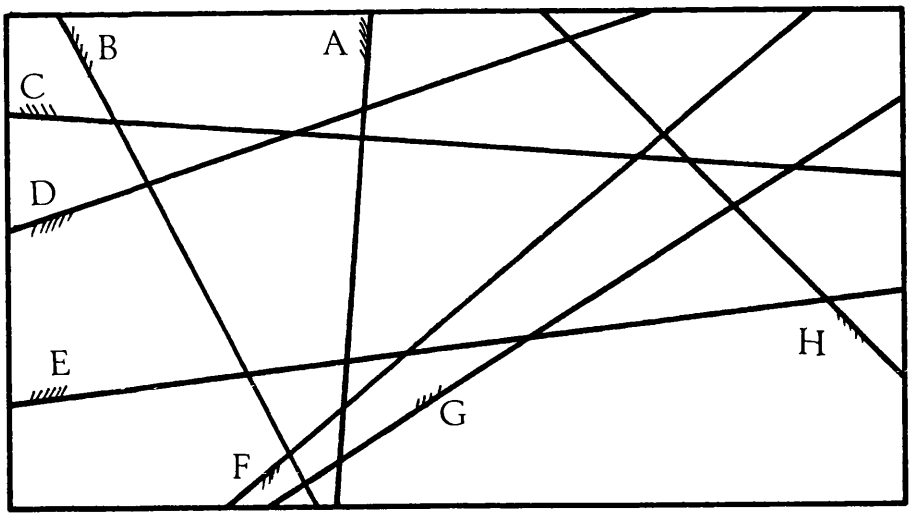
a. Surface types available are mainly the simple ones used to construct primitives. Researches on CSG techniques have mainly been concentrated on attacking assembly complexity and, relatively, have ignored surface complexity. Therefore, a lack of convenient means of handling sculptured and blend surfaces was common to many CSG modellers.

b. Lack of locality because each surface or primitive is defined globally (Fig.2.8). In locating a particular part of the object, the entire defining surface for that part has to be investigated to make sure that no part is missed out.

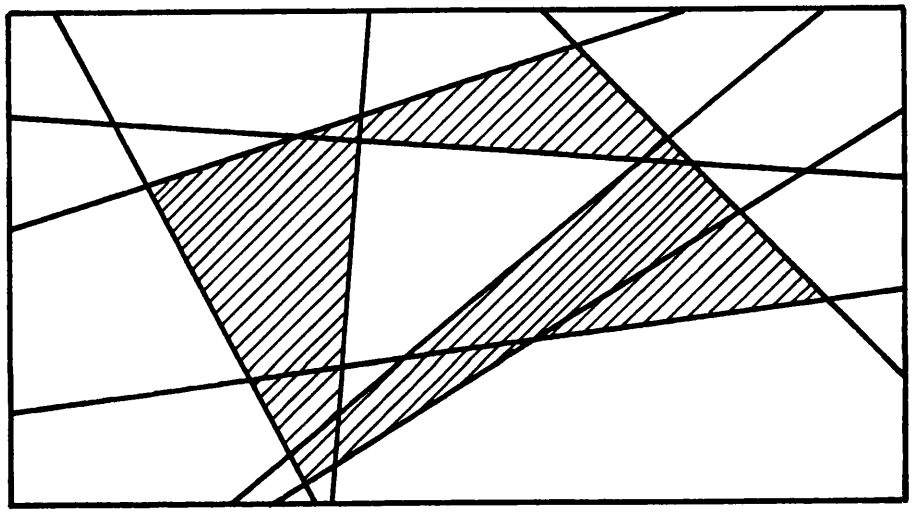
c. Redundant operations on the CSG definition tree. The model is usually defined as a set-theoretic operation tree (normally in the form of a reversed Polish expression) (Fig.2.8). Evaluation of a part of the model has to go through the entire tree.

d. Generating wire-frame views of the object is slow. The shape elements for a CSG model are primitives. A primitive is constructed from basic surfaces via set-theoretic operations. Primitives are related to each other via the same operation to form the final object. Vertices and edges which are needed for a wire-frame picture are not immediately ready for use in such a representation scheme. To generate such data, long computation times are required. This is inefficient for interactive operations.

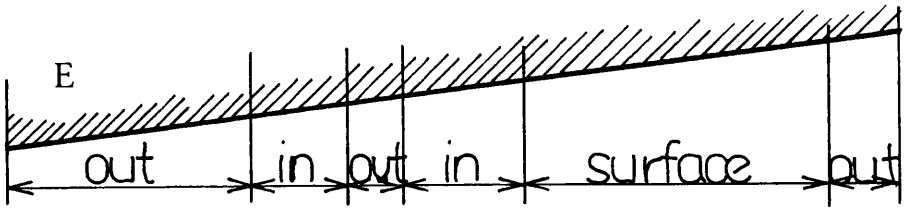
To use CSG as a powerful scheme of representation, these problems must be solved properly. A discussion on general methods for the solution of these problems is given here briefly. More detailed discussions on some special problems are given in later



(a)



(b)



(c)

$$B \cap D \cap H \cap (E \cup G) \cap (A \cup C \cup F)$$

(d)

fig. 2.8 lack of locality of a CSG model

chapters.

2.2.1 Primitives and Surfaces

The original idea of a CSG modeller was to construct a fairly complicated model using a small number of primitives. However, in practice there are vast number of objects with different shapes to be modelled, even in a particular area like mechanical engineering. It is difficult to cover all the shapes by only a few primitives. If the number of primitives increase, the range of different models that can be represented will increase. However, the number of types of primitives should not too big, or the modeller would become less compact and would lose the original and fundamental criteria of the CSG approach. When the number of primitive types is small, the shapes that the user can choose is rather limited, when only part of a primitive or a single face on the primitive is required to form a particular part of the model, the user has to use the entire primitive. This causes extra data to be stored and redundant operations to be performed (Fig.2.1c).

With a closer look at the construction of a model, we note that primitives are built up from basic surfaces via set-theoretic operations and, in turn, the model is constructed with these primitives also via set-theoretic operations. That is to say, internally the shape elements of a model are single surfaces, which are logically put together by set-theoretic operators to form the final model. This gives rise to the idea that the primitives can be reduced to single surfaces. This simplification greatly reduces many redundant operations and makes it possible to use surfaces economically.

As a CSG modeller does not explicitly store edges and vertices, surfaces are the basic operands in a model's construction and evaluation. This also means that the relationship between different surfaces are implicitly stored, and hence can be as topologically complicated as the geometry will allow. Therefore much more complicated surfaces can be used as long as they can be operated on by set-theoretic operators. This reduction from primitives to single surfaces is important if we wish to

expand the domain of solid modelling systems, specially in considering high order surfaces; for it gives the user the freedom to use a very large number of surfaces in constructing a model. This also effectively breaks the limitation on surface types and make a variety of surfaces available to be used in a CSG solid modeller.

One point should be mentioned here that this reduction is rather a generalisation on primitive types, not just a simplification. It is no restriction on the use of readily built primitives. In fact, the system can still provides a number of commonly used primitives and also allows the user to build up his own primitives for a particular application.

A surface that set-theoretic operations can be applied to is called a *half-space*. A half-space is a surface that divides space into inside (solid) and outside (air or empty) regions, with the surface in-between. Discussions on various half-spaces and techniques to define, represent and control them are given in Chapter 3.

2.2.2 Locality and Efficiency

Locality is a very important feature of a solid model. It affects the efficiency of its evaluation. The quicker a part of a model can be located, the quicker it can be evaluated, interrogated, or manipulated.

In a B-rep modeller, the locality is obtained from the adjacency relationship between faces, edges and vertices. These relationships depends on the data structure (the *winged edge* structure, for instance). By tracing along the linkage between these shape elements, a particular part of the model can be quickly located.

In a CSG modeller, since no topological adjacency relationship is stored, it is difficult to obtain locality in the same way. The basic shape elements are surfaces and the links are Boolean operators. As a surface is defined globally and most of them have infinite size, it is difficult to determine its position in space in the same way as a B-rep modeller locates an edge or a vertex. We have to attack this problem differently.

Generally speaking, the basic requirement for locality is to relate particular geometric information (surface definition) to particular positional information (coordinates). The closer and more accurately this is done, the quicker and more accurately the model can be interrogated.

In this thesis, attempts have been made to relate particular surfaces to particular regions in space. Unlike the adjacency relationship used in a B-rep modeller, what was used was a spatial relationship. This kind of relationship is established using *sub-division* and *pruning* techniques, which effectively solves the problems with locality and evaluation efficiency stated earlier. Detailed discussion on this topic and examples are given in Chapter 5.

2.2.3 Data for Pictures

Though the quick generation of wire-frame pictures of a modelled object still remains a problem with CSG solid modellers that accommodate curved surfaces, they are reasonably quick at generating section views (Wire-frame can be generated quickly from a CSG model made of planes [109]). They also favour ray casting techniques for generating realistic shaded pictures and other data needed in subsequent engineering applications, such as NC machining instruction generation.

As a research subject, CSG has not been paid as much attention as B-rep; as a representation scheme, it has not been used as widely as B-rep. But its precision conciseness and its capability of supporting various complicated surfaces make it more promising for more powerful, complete, efficient and sophisticated solid modeling systems.

CHAPTER 3

THE MATHEMATICAL DEFINITION OF COMPLICATED SURFACES AND BLENDS

Surface types are of crucial importance to a solid modelling system. They always determine its main features and its quality. In the system described in this thesis, surfaces and volumes are defined by implicit polynomial inequalities.

3.1 SOME SURFACE TYPES USED IN CURRENT SYSTEMS

Since the beginning of shape modelling using computers, a number of different surface types have been used [31]. Before any detailed discussion on implicit polynomials, we briefly review some surface types used in current systems. There are mainly the following surface types, which give different complexities and ranges of applications:

3.1.1 Planes

As we discussed in Chapter 1, planes have the simplest geometric form of all surface types. Because they are easy to represent and quick to manipulate, they are very commonly used in shape modelling. No matter how complicated a shape modelling system is, it always keeps planes as one of its surface types.

Mathematically, a plane can be defined in different ways and in different forms. In this thesis, the following form is used.

$$A_1X + A_2Y + A_3Z + A_4 = 0 \tag{3.1}$$

In shape modelling, the data needed to specify a plane can be given as A_1, A_2, A_3 *and* A_4, which are the coefficients of the above equation and, if normalised, they represent the direction cosines, $[A_1\ A_2\ A_3]$, of the plane and its distance from the origin, A_4. The data can also be given as the coordinates of three non-collinear points in space [Chapter 4]. This is sometimes more convenient for the user, specially, when planes are constructed from data obtained by measuring or triangulation. Usually, they are converted to the normalised implicit form, because it requires the least data possible, and is also the most numerically stable[31].

Some solid modelling systems (VOLE and DORA, for instance[109]) use only planes as their surface type. The advantage of this is that pictures can be quickly produced. The

main limitation of planes is that they are not really satisfactory for modelling curved surfaces. Though theoretically they can represent all kinds of curved surfaces by approximation, practically this needs enormous amount of data that quite often exceeds the capacity of the modeller and slows down evaluation, which reduces their advantages mentioned above.

3.1.2 Cylinders, Spheres, Cones and Tori

This group of surfaces have obvious advantages over planes in representing objects with curved surfaces. This is especially true in mechanical engineering, where many components are produced by machining on machine tools. These machined components are usually made of surfaces belonging to this group (Fig.2.5). A solid modelling system which employs such surfaces is capable of modelling most of the machined components in mechanical engineering.

The mathematical definitions of these surfaces are relatively simple, though they are more complicated than that of planes. They can be defined in either implicit or parametric form. For example, the implicit form of a cone is

$$\frac{X^2}{a^2} + \frac{Y^2}{a^2} - \frac{Z^2}{c^2} = 0 \tag{3.2}$$

and its equivalent in parametric form is

$$X = au\cos(v) \tag{3.3a}$$
$$Y = au\sin(v) \tag{3.3b}$$
$$Z = bu \tag{3.3c}$$

Their construction by computer is more difficult than that of planes, because their definitions are affected by their positions and orientations in space. They have the simplest form when their central lines are parallel to one of the axes with the apex at the origin, as in the equation shown above. If they are to be stored in the simplest form, transformation techniques are needed to apply them to the construction of an object. Sometimes both of the

implicit and parametric forms are used to speed up certain kinds of calculations, for example, when the intersection between surfaces is to be calculated.

Since this group represents the most popular surface types and covers most of the machined components in mechanical engineering, many systems use this group of surfaces (including planes) as their only surface types. Special algorithms can be designed to handle such surfaces quickly. For most of their intended applications, they are satisfactory, but further extension on the range of their applications is quite limited. Because of the method used in implementing systems using spheres, cones, etc, it is difficult to accommodate more complicated surface types without seriously changing data structures and algorithms.

3.1.3 General Quadrics and Cubics

This group of surfaces is more general but more difficult to specify and manipulate. The main purpose of their use is to represent some irregular surfaces which are difficult for those discussed above.

Though they can be defined in implicit form, they are usually specified in parametric form in practice. Their equations are usually formed by interpolating over points, which are obtained or specified beforehand.

3.1.4 Swept Surfaces

A swept surface is formed by moving a space curve along another space curve, or rotating a space curve about a line. Sometimes, moving and rotation are used simultaneously to form more complicated surfaces. If a surface is used in place of the moving curve, the volume and surface of a complicated object can be generated at the same time. The latter is very suitable for a CSG solid modeller, as volume and surface are dealt with consistently (Fig.3.1). The shapes of those engineering components that are produced on a lathe can be easily constructed in this way (Fig.2.5).

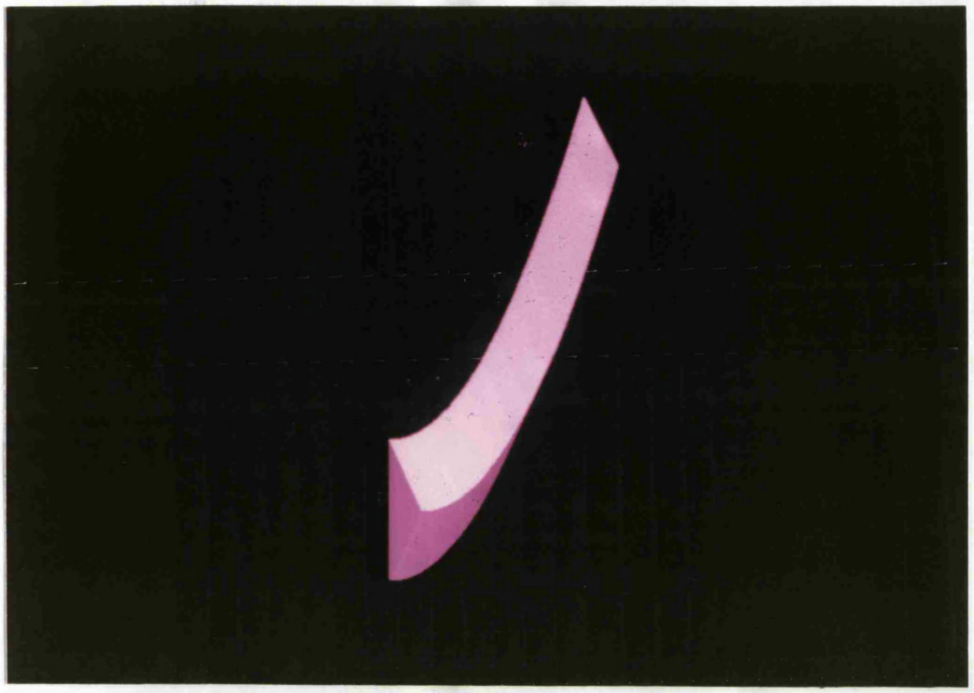


Fig.3.1 A Solid by Sweeping A triangle Along a Quadratic Curve

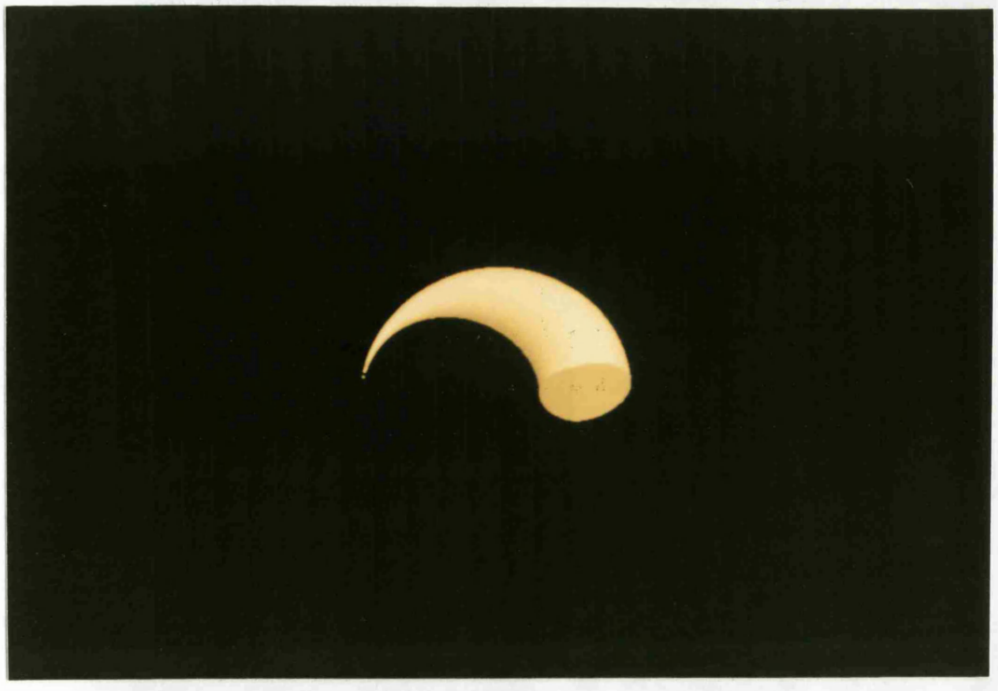


Fig. 3.2 An Arbitrary Cone Defined Using the Multi-Stage Method

The curves and surfaces used in generating a swept surface can be any of those surface types discussed earlier, (curves can be their two-dimensional equivalent). The difficulty in generating this kind of surfaces is in the control of the sweeping movement and the consistency of the generated surface. For example, if a curve penetrates the surface it has been generating, the result would be ambiguous if it were used to define a solid volume.

Swept surfaces can be very complicated, and they may be difficult or even impossible to generate using other techniques. Many shape modelling systems use sweeping techniques to form their surfaces or volumes. They are particularly suitable and convenient when constructing a model interactively.

3.1.5 Sculptured Surfaces

These kinds of surfaces are the most complicated surface type presently used in shape modelling. Their applications are mainly in the design and manufacturing of aircraft fuselages, car bodies and ship hulls. They can also be used to represent objects in other areas, such as those in mechanical engineering [20]. Recently, they have been used to model human organs, faces, and other natural objects. They have also found their applications in computer animation for film making and TV programs.

A sculptured surface is usually defined in parametric form. It is very difficult to specify a sculptured surface of practical use with one single equation. They are usually formed by several pieces of surfaces, or *patches*, which are defined independently, though appropriate restrictions are imposed to make these patches consistent at their joints. This also explains the reason for using the parametric form for sculptured surfaces: it is much easier to carry out local changes or modifications on a parametric patch than on other surface types.

A parametric patch is in fact an extension of parametric curves from the two-dimensional plane to three-dimensional space. In geometric modeling, an parametric curve is usually an interpolation over a set of points, which are called *control points*. If we

interpolate over a two-dimensional grid, we obtain a patch. In most cases, a parametric patch is defined over a topologically rectangular area, and is interpolated along each row and column. Hence, two variables are needed as parameters to control this patch. A parametric patch of two variables is called a *bi-parametric* patch. Bi-parametric patches have been widely used to form many sculptured surfaces.

Ferguson's cubic surface patches, Bezier UNISURF surface patches, Coon's patches and B-spline patches are some basic parametric patches current used in geometric modelling. There are some other parametric surface patches, such as Tensor-product (or Cartesian product) surfaces, rational parametric surfaces and parametric spline surfaces and so on.

These patches can be used to represent a single surface or, with certain modification, can be used to form a composite surface in a piecewise manner. B-spline patches are considered to be the best of them for defining a composite surface with satisfactory continuity at junctions between each patch.

For a more detailed discussion on parametric surfaces, interested readers are referred to references [14][30][35][31].

3.1.6 Blend Surfaces

Blend surfaces have recently attracted the attention of many engineers. Due to the difficulty in defining and handling them, and also because of the fact that they are usually not the basic surfaces for the majority of engineering components, they have long been ignored in geometric modelling. As computer technology has progressed and shape modelling techniques have developed, it is now possible to handle these as one class of surfaces.

Though they are usually not the critical and basic surface types (such as cylinders or spheres), they are non-ignorable parts of many engineering components. Without them, a solid modelling system in mechanical engineering can hardly be considered as complete. The handling of blend surfaces also implies the handling of simpler surfaces used in engineering.

A blend surface is more complicated in the sense that it has more constraints and has to deal with more complicated situations. It is more difficult to define using existing techniques that work well for sculptured surfaces.

Blend surface definition, manipulation, integration with other surface types and applications in modelling engineering components are the main task of this thesis. More will be said about blend surfaces in the rest of this chapter and later chapters.

3.2 IMPLICIT POLYNOMIAL SURFACES

Generally speaking, Parametric surfaces have been dominant in the field of geometric modelling. Different types of parametric surfaces and different techniques for handling them have been developed and used in most of the existing systems.

Though parametric patches have been widely used to define sculptured surfaces, they are not the best in all circumstances. In this thesis, the aim is to develop a convenient technique for representing blend surfaces which should naturally include other surfaces being blended. It is difficult to define a blend surface between two or more parametric patches using current techniques, as was pointed out in Chapter 1.

In this thesis, implicit polynomials have been chosen to define both simple and complicated surfaces, and particularly blend surfaces that exist on many engineering components, but are unfortunately missing from their solid models generated by many existing solid modellers.

An implicit polynomial is in fact a penalty or scale function [82], which gives each point in space a value. All the points that have the value of 0 constitute the surface represented by the polynomial.

Theoretically, implicit polynomials are capable of representing many types of simple and complicated surfaces in any position and orientation in space. Theory and techniques for dealing with low order polynomials (not higher than 3) are well developed [31]. They are easy to understand and to use. However, it is rather difficult to construct a required complicated surface, such as a blend between two pipes, using polynomials in the same way as a simple one is defined. It is difficult even to define univariate equations, not to mention multivariate high order surfaces, such as those needed in engineering. This is why whenever a complicated surface is required, people tend to use parametric techniques to approximate it with several parametric patches in a piecewise manner.

The difficulties in using polynomials to represent high order surfaces are mainly in the specification of the actual equation and the control of the shape, size and orientation of the surface. Whether polynomials can be adequately and widely used in solid modelling systems depends on how well these problems can be solved. Some methods are discussed in this chapter to attack these problems.

To make the discussion easier, it is necessary, before detailed description of the method, to introduce the idea from which the author's techniques were developed.

3.2.1 Liming's Equation for Quadratic Curves

In aircraft fuselage design, Liming [60][31] used straight line equations as elements to construct implicit quadratic curves. For example, a quadratic curve, C, can be defined in the following form

$$C : (1-\lambda)PQ + \lambda RS = 0 \tag{3.4}$$

where P, Q, R and S are the linear functions

$$\begin{aligned} P &= A_p X + B_p Y + C_p \\ Q &= A_q X + B_q Y + C_q \\ R &= A_r X + B_r Y + C_r \\ S &= A_s X + B_s Y + C_s \end{aligned}$$

and λ lies in the range [0,1].

Equation (3.4) is equivalent to a general quadratic and has the property that it passes through all the points where P or Q meets R or S (Fig.3.3).

If line S is moved towards line R, eventually the lines will merge (Fig.3.4). When S and R become identical, equation (3.4) becomes

$$(1-\lambda)PQ + \lambda R^2 = 0 \tag{3.5}$$

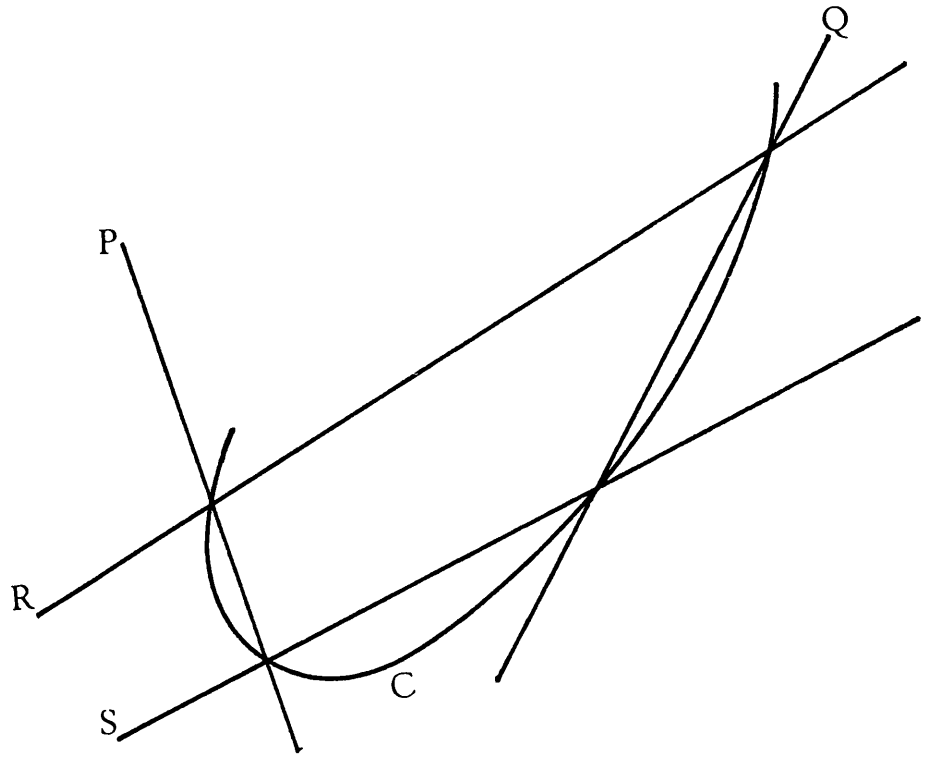


Fig.3.3 Liming's Quadratic Curve (1)

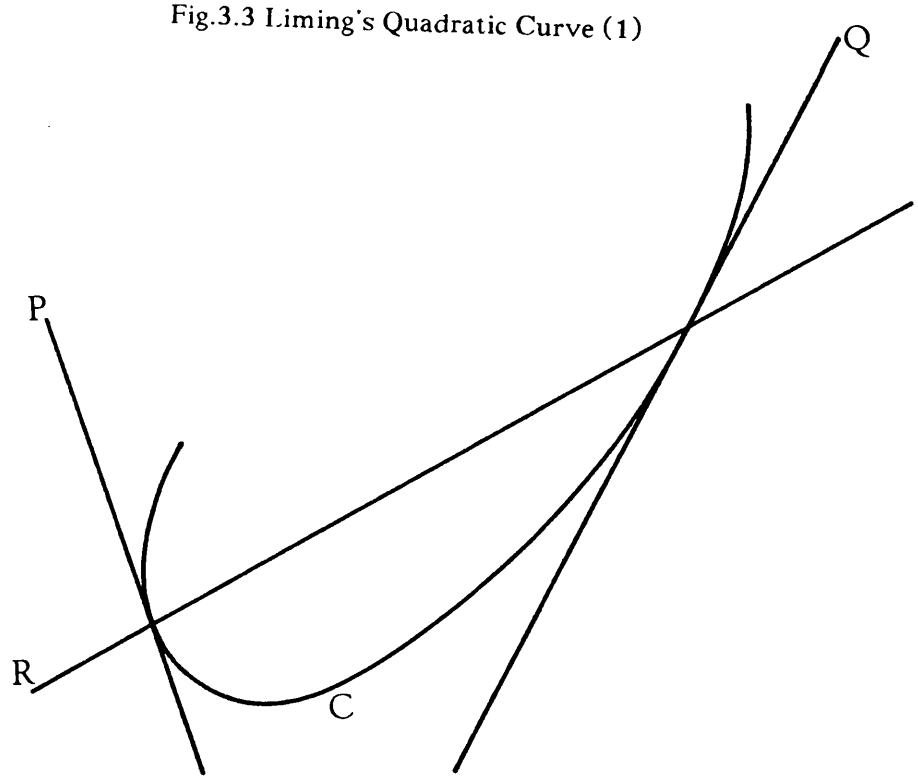


Fig.3.4 Liming's Quadratic Curve (2)

This change from (3.4) to (3.5) is very important. It is clear from Fig.3.3, 3.4, equation (3.4) and (3.5) that

a. In (3.4), four lines are used to define the quadratic, but in (3.5) only three are needed, which is the smallest number possible for a non-degenerate curve.
b. The four lines in (3.4) have equal importance in the definition of the quadratic, whereas the three lines in (3.5) have different contributions: lines P and Q become two tangent lines of the quadratic curve.

The property of this method that the defined curve can be controlled so as to be tangent to the defining straight lines makes it a simple and effective way to define an implicit polynomial which links another two or more together smoothly. The defining curves need not be straight lines: any polynomial will do, hence we have a much more convenient and reliable way for defining blend curves in the two-dimensional plane and this makes it a potential method to define blend surfaces in three-dimensional space.

Conventionally, apart from some well known curves like a circle or a parabola, a general implicit polynomial equation is constructed by choosing its coefficients explicitly or by requiring it to pass through certain points in space. While this is satisfactory for low order polynomials, it is difficult or even practically impossible to use for high order polynomials.

Liming's method of constructing a quadratic shows that there is another way to solve the definition problem. We do not have to build up an implicit polynomial directly from coefficients or coordinates in one step; we can use defined polynomials to define other more complicated ones in a rather simple and intuitive way. Furthermore, using this method, we have better understanding and control over what we are constructing. We know what shape it should be and where it should be. If the result goes wrong, we know how to check and

correct it.

3.2.2 Extension of Liming's Method in Three-Dimensional Space

Liming's method is convenient and effective, but some improvements and extension are necessary to make it practically useful in solid modelling systems.

a. By replacing straight lines in the two-dimensional plane with planes in three-dimensional space, a quadric surface can be defined in any position and orientation in space (Fig.3.5).
b. By changing the number of planes, their relative positions and orientations, more complicated surfaces can be defined that might be difficult to specify in other conventional ways (Fig.3.6).
c. By replacing some of the planes with higher order surfaces, or just with constant values, polynomials of different shape and order can be defined just as easily (Fig.3.7).
d. To expand further the surface types, use more steps, that is to say, a surface defined in one equation can be used in another equation to define a more complicated surface. In this multistage way, much more complicated surfaces can be obtained (Fig.3.8).

These extensions make this technique much more powerful and versatile, and they still keep the property of the method that the defining surfaces and the defined surfaces can be jointed together smoothly. Therefore, it is not only a useful method of constructing high order polynomials, but also a effective way of defining blend surfaces, which is not an easy job for parametric surfaces.

For clarity and consistency, we now define some notation that will be used for the remainder of this thesis.

The basic elements used by this method are planar surfaces and constants. A planar surface is defined as

$$H_{1a} = A_1X + A_2Y + A_3Z + A_4 = 0 \qquad (3.6a)$$

or

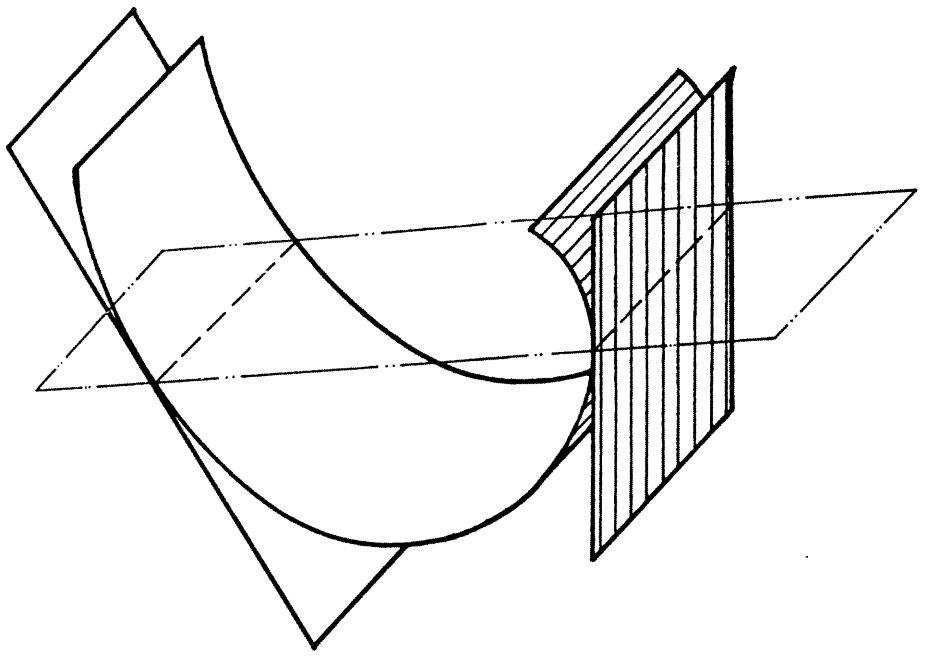


fig.3.5 a quadric defined using Liming's method

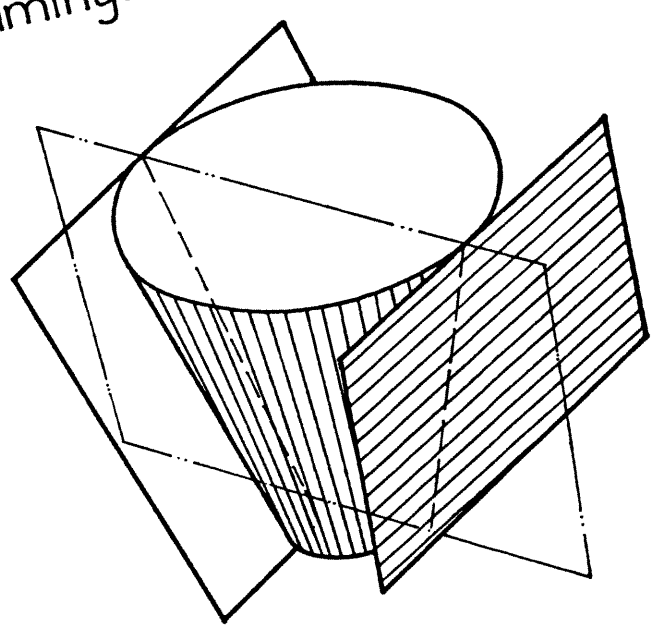


fig.3.6 changing shape of a quadric by changing positions and orientations of the planes

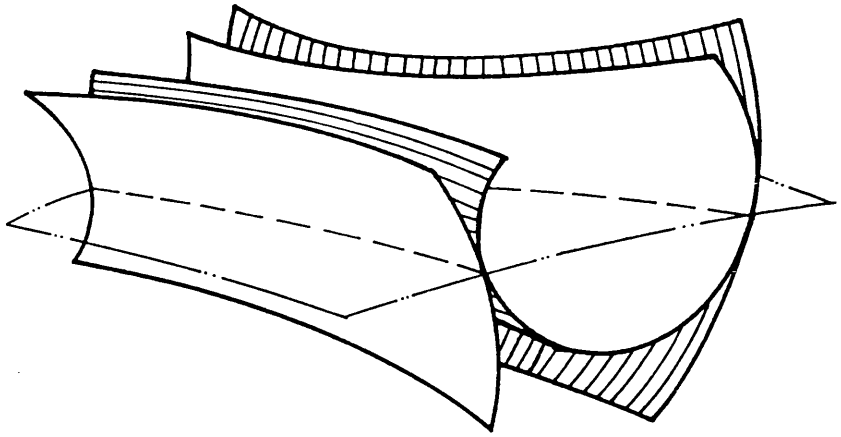


fig.3.7 complicated surface using curved defining surfaces

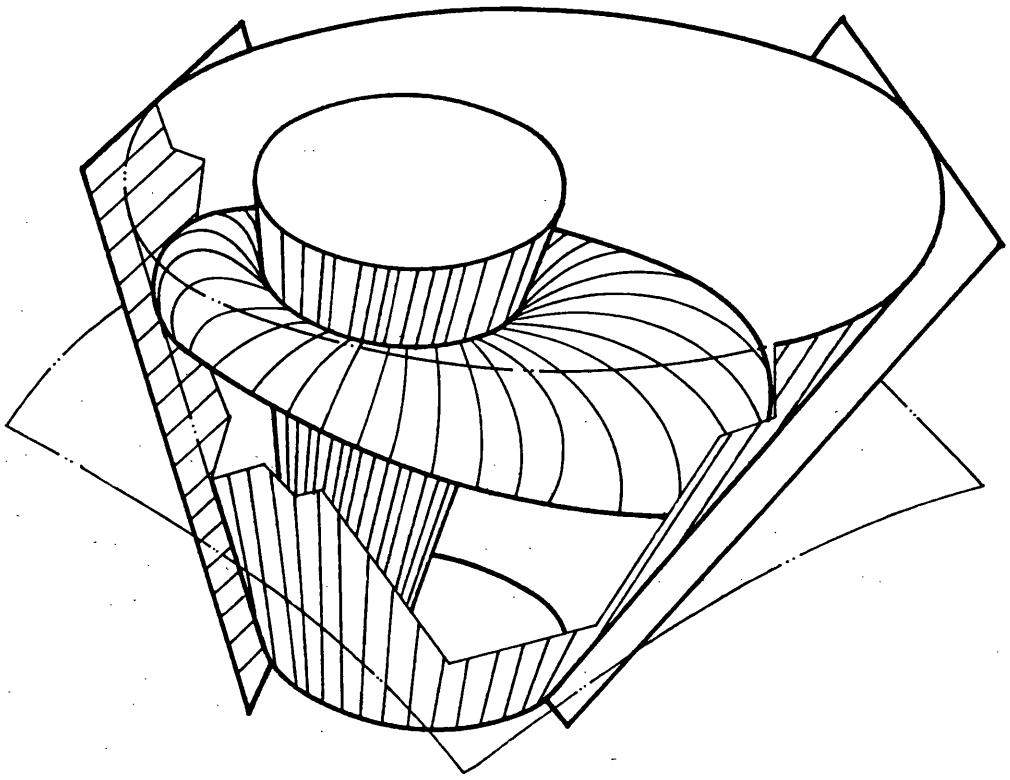


fig.3.8 multistage method for high order polinomials

$$H_{1b} = B_1X + B_2Y + B_3Z + B_4 = 0 \qquad (3.6b)$$

where A_1, A_2 and A_3, or B_1, B_2 and B_3 are usually normalised (This is done automatically by all the author's software.) and therefor A_4 or B_4 represents the distance between the plane and the origin.

To form a general notation for a surface, we assume that a half-space, planar or curved, is named as H_{ij}, where i is an integer number indicating the order of the half-space and j is a letter which distinguishes a particular surface from the rest. For example, a plane can be named as H_{1a}, H_{1b}, or H_{1p}; a quadric can be named as H_{2a}, H_{2c} or H_{2r}. (This should not be confused with the names used in an input language [Chapter 4], where, of course, a surface can have a more sensible name such as CYLINDER1, TORUS, etc.)

Constants are usually represented by letters R, P or D, and control constants are normally represented by letters λ and γ.

The surface normal and gradient are represented by letter N and G respectively, for example,

$$N_{3a} = [N_{3xa} \quad N_{3ya} \quad N_{3za}]$$

and

$$G_{5a} = [G_{5xa} \quad G_{5ya} \quad G_{5za}]$$

Surface normals are for points that lie on the surface, while gradient can be for a point anywhere in space.

3.3 CONSTRUCTION OF COMPLICATED IMPLICIT POLYNOMIALS FROM SIMPLE ONES

The basic idea of the method proposed here is to define a higher order surface in a multistage manner using planar half-spaces and constants as the elements from which it is assembled. To make the specification easier, certain transformation techniques are used to move half-spaces around whenever required.

3.3.1 Cylindrical and Spherical Half-Spaces

A cylinder or sphere can be defined directly from its shape, size and orientation. For example, the equation

$$X^2 + Y^2 - 100 = 0 \tag{3.7}$$

defines a cylinder of radius 10, with the Z axis as its central line. But for a general cylindrical or spherical surface, such as the above cylinder in an arbitrary position and orientation in space, the equation would have to take the following form

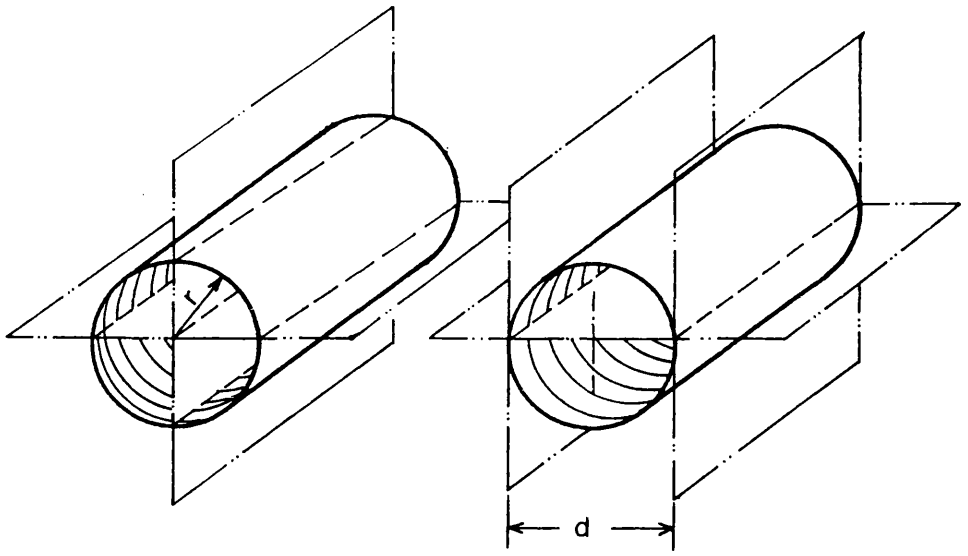
$$AX^2 + BY^2 + CZ^2 + DXY + EYZ + FXZ + GX + HY + IZ + J = 0 \tag{3.8}$$

which is neither intuitive nor easy to use.

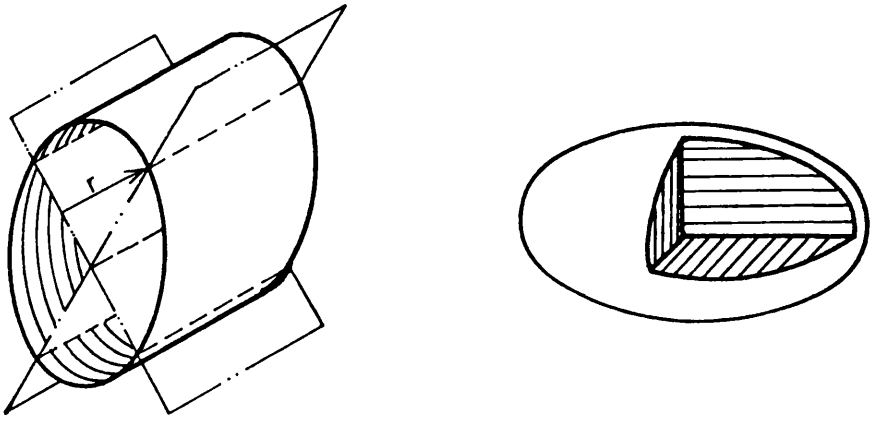
However, use the multistage method, an arbitrary cylinder may be defined as

$$H_{2c} = H_{1a}^2 + H_{1b}^2 - R^2 = 0 \tag{3.9a}$$
$$H_{1a} = A_1X + A_2Y + A_3Z + A_4 \tag{3.9b}$$
$$H_{1b} = B_1X + B_2Y + B_3Z + B_4 \tag{3.9.c}$$

where H_{1a} and H_{1b} are two normalised planar half-spaces perpendicular to each other. R is the radius of the cylinder. The intersection line of H_{1a} and H_{1b} is the central line of the cylinder. Since the intersection line can be in any orientation in space, so can the central line of the cylinder.[Fig.3.9a] This form is clearer and simpler to understand and use. Simi-



(a) (b)



(c) (d)

fig.3.9 cylindrical and spherical half-spaces

larly, a sphere may be defined as

$$H_{1a}^2 + H_{1b}^2 + H_{1c}^2 - R^2 = 0 \tag{3.10}$$

where H_{1a}, H_{1b} and H_{1c} are planes perpendicular to each other. Their intersection point is the center of the sphere and R is its radius.

The constant R here is not necessary and can be replaced with another planar half-space, if the user did not wish to use a constant or the radius of a cylinder or sphere is not known at the time of its construction. For example, the cylinder defined earlier can be specified as

$$H_{1c}^2 - H_{1a} H_{1b} = 0 \tag{3.11}$$

where H_{1a} and H_{1b} are parallel to each other, while H_{1c} is perpendicular to both of them. The distance between H_{1a} and H_{1b} is the diameter of the cylinder. The central line of the cylinder lies in H_{1c} and parallel to H_{1a} and H_{1b} (Fig.3.9b).

If the positions of the planes and the angles between them are changed, or the value of the radius is changed, or each plane is multiplied by a factor to modify its potential value, and then different cylindrical and spherical half-spaces can be defined. They have different cross sections and appearances (Fig.3.9c,9d).

3.3.2 Conic Half-Spaces

Cones are another type of basic shapes used in mechanical engineering. The equation

$$\frac{X^2}{A^2} + \frac{Y^2}{A^2} - \frac{Z^2}{C^2} = 0 \tag{3.12}$$

defines a cone with the origin as its apex and Z axis as its central line. This is obvious only in such a particular situation. The equation form of a general cone in an arbitrary position and orientation is not so easy to specify.

With the multistage method, however, this becomes much simpler. By studying the equation of the cone just given above, we notice that there is a special relationship between the cone and the axes (as described above) which makes the particular definition simpler. Therefore, if we can establish this special situation somewhere else we can define a cone in a similar way. This is what the multistage method can do easily, because three planar surfaces can easily be positioned to be perpendicular to each other (as we did in defining a sphere), and this resembles the situation as at the origin. For a cone like in Fig.3.10, the equations can be

$$H_{1a}^2 + H_{1b}^2 - \lambda_c H_{1c}^2 = 0 \tag{3.13}$$

where H_{1a}, H_{1b} and H_{1c} are three planes perpendicular to one another to resemble the structure of the Cartesian coordinates but in a different position and orientation. The value of λ_c is to control the angle of the cone.

If a cross section other than a circle is required, the equation can be written as

$$\lambda_a H_{1a}^2 + \lambda_b H_{1b}^2 - \lambda_c H_{1c}^2 = 0 \tag{3.14}$$

with different value of λ_a, λ_b and λ_c, the shape of the cross section and the angle can be controlled, allowing elliptical cones to be easily specified. To make this group of surfaces more general, curved surfaces can be used in position of planes, for example, by using two cylindrical surfaces in this equation, a complicated surfaces can be defined, as shown in (Fig.3.2).

3.3.3 Toroidal Half-Spaces

The torus is another popular shape used in engineering. For example, the fillet between two parallel co-axial cylinders and the shape of a hand-wheel are tori.

A torus in a particular position with its centre at the origin and the central circle curve lying in the $X-Z$ plane may be defined as

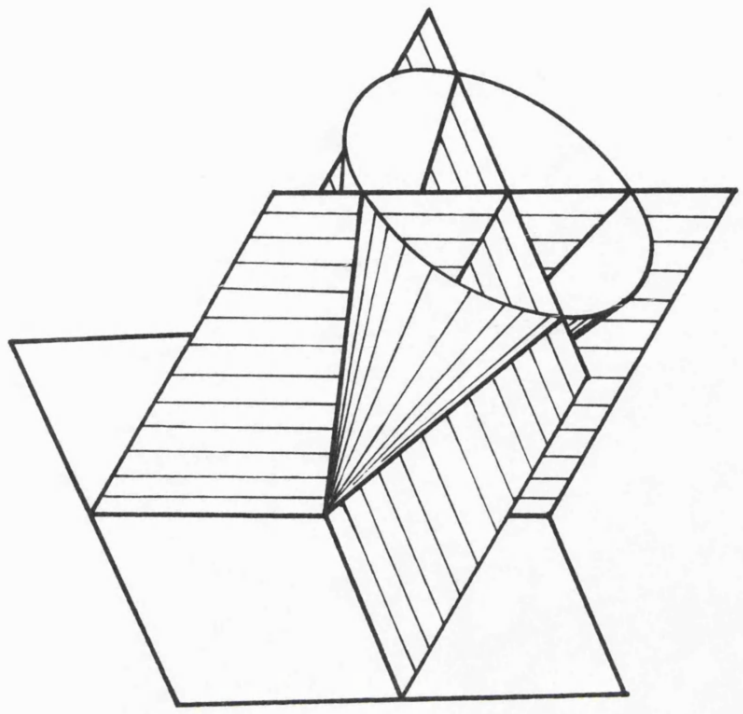


fig.3.10 a cone in an arbitrary orientation

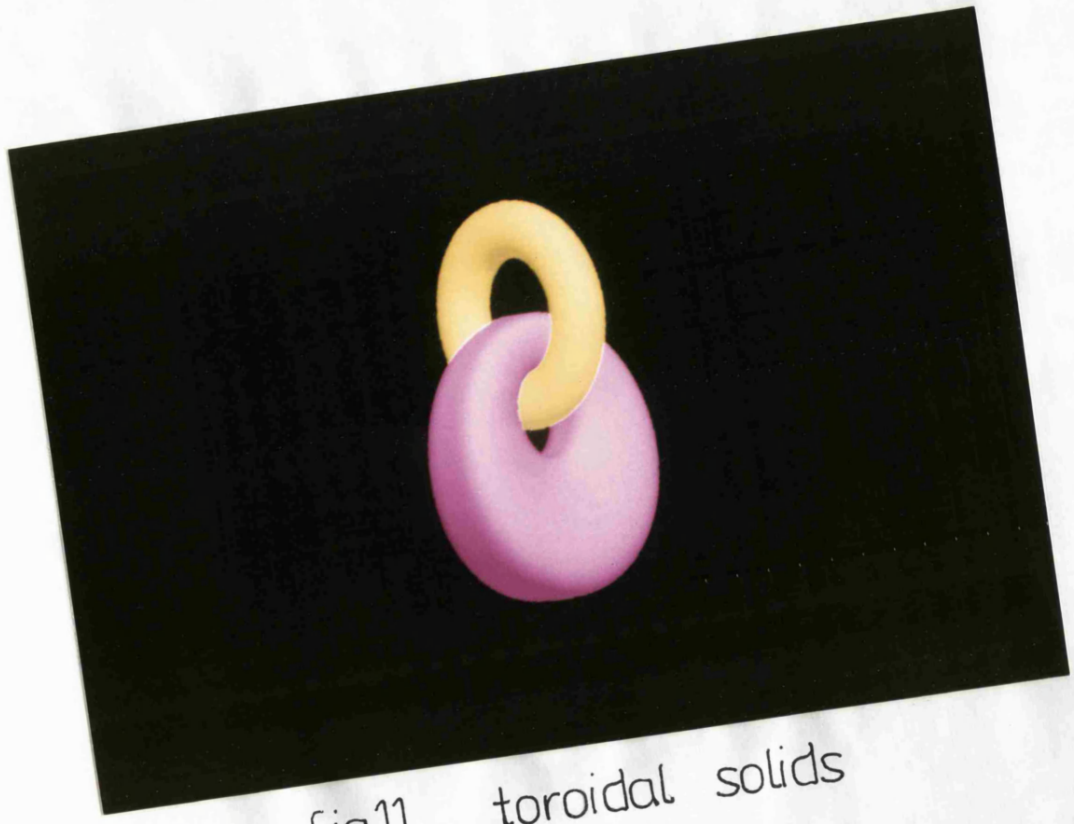


fig.11 toroidal solids

$$(\sqrt{X^2+Z^2} - R_1)^2 + Y^2 - R_2^2 = 0 \tag{3.15}$$

where R_1 is the radius of the central circle curve and R_2 is the radius of the cross section circle of the torus. But a torus in a arbitrary position and orientation is rather difficult to define in the same way.

It is much more straightforward and clearer if the multistage method is used. For example, a cylinder can be easily formed in the way that we used earlier and a plane can be placed to intersect the cylinder at right angles to form a circular space curve, which can then be used as the central curve of the torus. The definition may have the form

$$H_{2a}^2 + H_{1c}^2 - R_2^2 = 0 \tag{3.16a}$$

$$H_{2a} = \sqrt{H_{1a}^2 + H_{1b}^2} - R_1 \tag{3.16b}$$

where R_1 is the radius of the central circle curve and R_2 is the radius of the cross section circle of the torus.

The operation of the square root is expensive on most computers and it also complicates the form of the torus representation. To simplify this, equation (3.16) can be rewritten as

$$4R_1^2(H_{1a}^2 + H_{1b}^2) - (H_{1a}^2 + H_{1b}^2 + H_{1c}^2 + R_1^2 - R_2^2)^2 = 0 \tag{3.17}$$

This can be done automatically by a program which keeps the former form at the user definition stage because of its intuitiveness.

By altering of the number, position and orientation of the defining half-spaces, more shapes in the group of topologically toroidal surfaces can be formed, as shown in Fig.3.11.

3.3.4 Half-Spaces by Off-Setting

Other irregular half-spaces can be defined by taking an off-set from two or more other half-spaces which are multiplied together. Using this technique, we can define a half-space similar to the composite surface shape of the defining half-spaces, but with all the

sharp corners smoothed.

For example, if H_{2a} and H_{2b} are two cylinders, a surface close to their composite shape can be defined using equation

$$H_{2a}H_{2b} - P = 0 \tag{3.18}$$

where P is a user defined constant, which determines how close the defined surface will be to the defining surfaces. Fig.3.12 shows the defining cylinders and the defined surface of this example. Figure 3.13 shows another example, where a cube with its sharp corners and edges smoothed is defined by six planes (The defined surface is not singular, non-interesting parts are chopped off by set-theoretic operations).

A great number of different shapes can be defined by using different shapes, different numbers of defining surfaces and by changing their relative positions.

3.3.5 Half-Spaces by Balancing

Though the off-setting technique is a simple way to define complicated surfaces, sometimes it is difficult to decide the value of the off-set. The balancing technique, on the other hand, is convenient to use in this respect, but more half-spaces are needed.

The basic idea of generating a half-space by balancing is to define two groups of half-spaces, and by changing the value of λ, a half-space lying between them can be generated.

For example, if H_{ia}, H_{jb}, H_{kc} and H_{ld} are four (the number can be different) half-spaces, then the equation

$$(1-\lambda)H_{ia}H_{jb} - \lambda H_{kc}H_{ld} = 0 \tag{3.19}$$

defines a surface between $H_{ia}H_{jb}$ and $H_{kc}H_{ld}$. The value of λ is used to control the relative position of the defined surface between the two groups of defining surfaces. As an example, the curved surface in Fig.3.14 is generated between three cylinders and a plane.

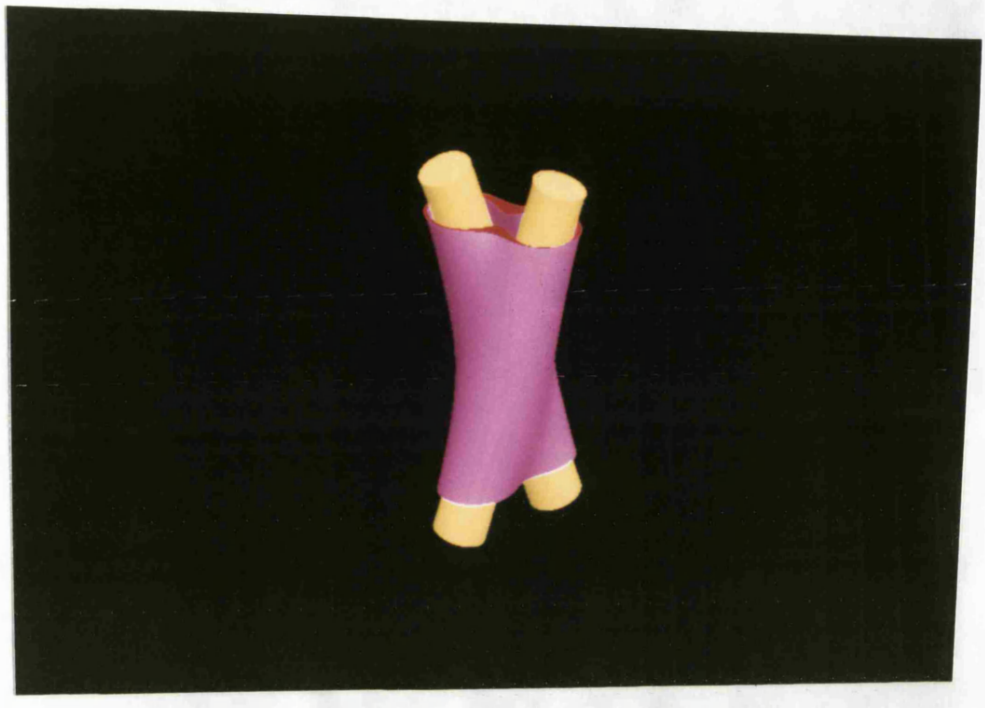


Fig.3.12 A Surface by Offsetting Two Cylinders

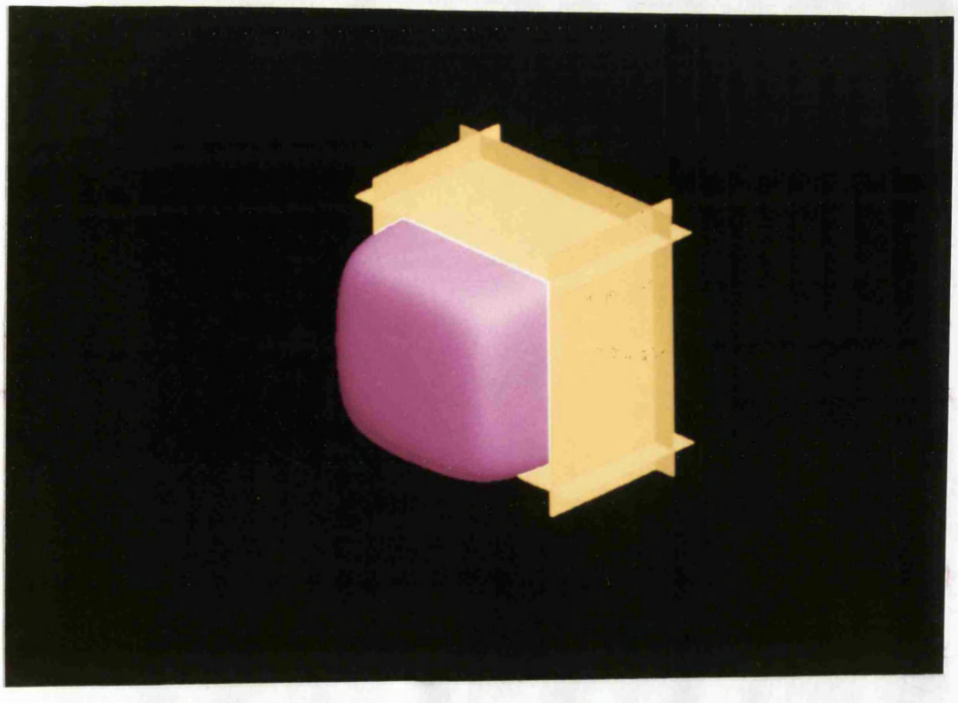


Fig.3.13 A Surface by Offsetting a Box

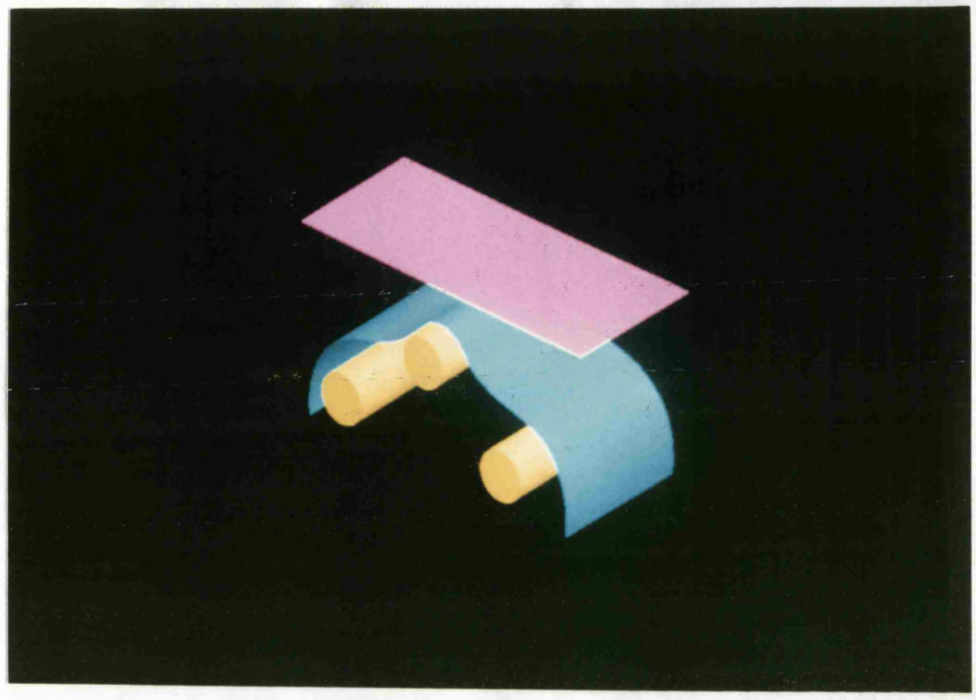


Fig.3.14 A Surface by Balancing Three Cylinders and a Plane

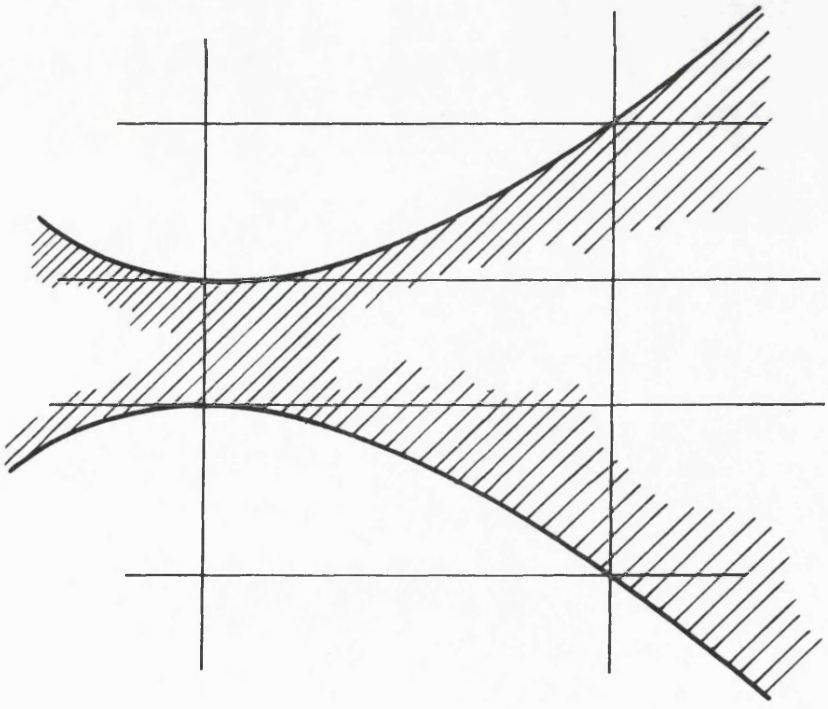


Fig.3.15 The Neck of the Hip-Joint Replacement (Fig.3.27)
by Balancing Two Group Surfaces

A wider range of half-spaces can be defined than by the off-setting method, because the defined half-space is allowed to pass through some intersection of the defining half-spaces, as the surface at the neck of the hip-joint (Fig.3.27) was defined (Fig.3.15).

3.3.6 Other Complicated Surfaces

It is difficult to list all the methods that can be used to define half-spaces in the multistage way. In practice, different methods always come from applications. However no matter how many cases there are, the multistage method can potentially deal with almost all the requirements for representing engineering components and is no more difficult than other methods to use, and, in many common circumstances, is substantially easier.

3.4 TECHNIQUES FOR DEFINING CORNER BLEND SURFACES

The multistage method is very suitable for defining blend surfaces, for the tangential relationship between basic surfaces and blend surfaces is easy to establish and control.

Corner blend surfaces exist on many engineering components. They distinguish themselves in that they link other surfaces, which are called *blended surfaces* in this thesis, with only one bend for their cross-section shape; in other words, since the blended surfaces lie on the same side of the blend surface, in principle, they can be dealt with by techniques such as those which try to reproduce the effect of rolling a ball along the joins between the blended surfaces. (This particular technique is hard, in practice, to implement.) Chamfers, fillets, and blends between intersected cylinders, the blend at an edge formed by two surfaces or at a corner of three or more surfaces, are all of this type [44][45][46][47][63][105][107].

3.4.1 Basic Idea of Generating and Controlling Corner Blend Surfaces

Blend surfaces are usually more complicated than the surfaces to be blended. Commonly used blend surfaces are those between simple solid primitives, such as cylinders, cones and tori. Combination of these primitives are enormous, and furthermore, blend between complicated shape or blend on previous blend are also very often needed in modelling mechanical engineering components. However, in most of the cases where corner blends are needed, the basic idea of defining them remains the same. The idea behind all the corner blend surface definitions in this thesis is the extended Liming's method, which was described earlier.

Here, for the definition of corner blend surfaces, we introduce the basic technique once more, but in a more general way. Suppose that H_{ia} and H_{jb} are two surfaces to be blended and H_{kc} is another surface intersecting both H_{ia} and H_{jb}, then the following equation

$$H_{ld} : (1-\lambda)H_{ia}H_{jb} - \lambda H_{kc}^2 = 0 \qquad (3.20)$$

defines a surface H_{ld}, which is tangent at the intersection curves between the two surfaces H_{ia}, H_{jb} and the other surface H_{kc} so that surface H_{ld} links H_{ia} and H_{jb} together smoothly and can be treated as a blend between them. λ is a constant in the range [0, 1] [Fig.3.16].

A corner blend surface can be very easily defined using this method. Using this method, we can also easily control the shape and size of the blend surface.

In order to define a blend between H_{ia} and H_{jb}, we must decide which areas on the blended surfaces are to be replaced by the blend. These areas can be easily controlled by surface H_{kc}, which is called the *ranging surface*. Using different surfaces as the ranging surface, and changing its relative position to the blended surfaces, the blended areas can be controlled to meet the requirement conveniently, as shown in Fig.3.17.

Having decided the areas to be replaced by blend surfaces, we also have to control the relative position or the *extent* of the blend surfaces over these areas. This is effectively controlled by the constant λ. When λ is 0, the blend surface appears the same as the blended surfaces, in other words, no blend is generated; when λ is 1, the blend takes the shape of the ranging surface and the blended surfaces are linked together directly be the ranging surface; when λ is between 0 and 1, it has a different shape and lies between the blended surfaces and the ranging surface. As λ becomes greater, the blend becomes fatter and fatter, as shown in Fig.3.18.

To use the generated blend surface on the model, the un-wanted part of it has to be chopped off. This is done by using set-theoretic operations. The intersection of the blended surfaces and the ranging surface provides us with a natural boundary of the blend surface. Chopping the blend surface along this curve, we then obtain just what is needed. The set-theoretic definition of the result of the blending can be written as

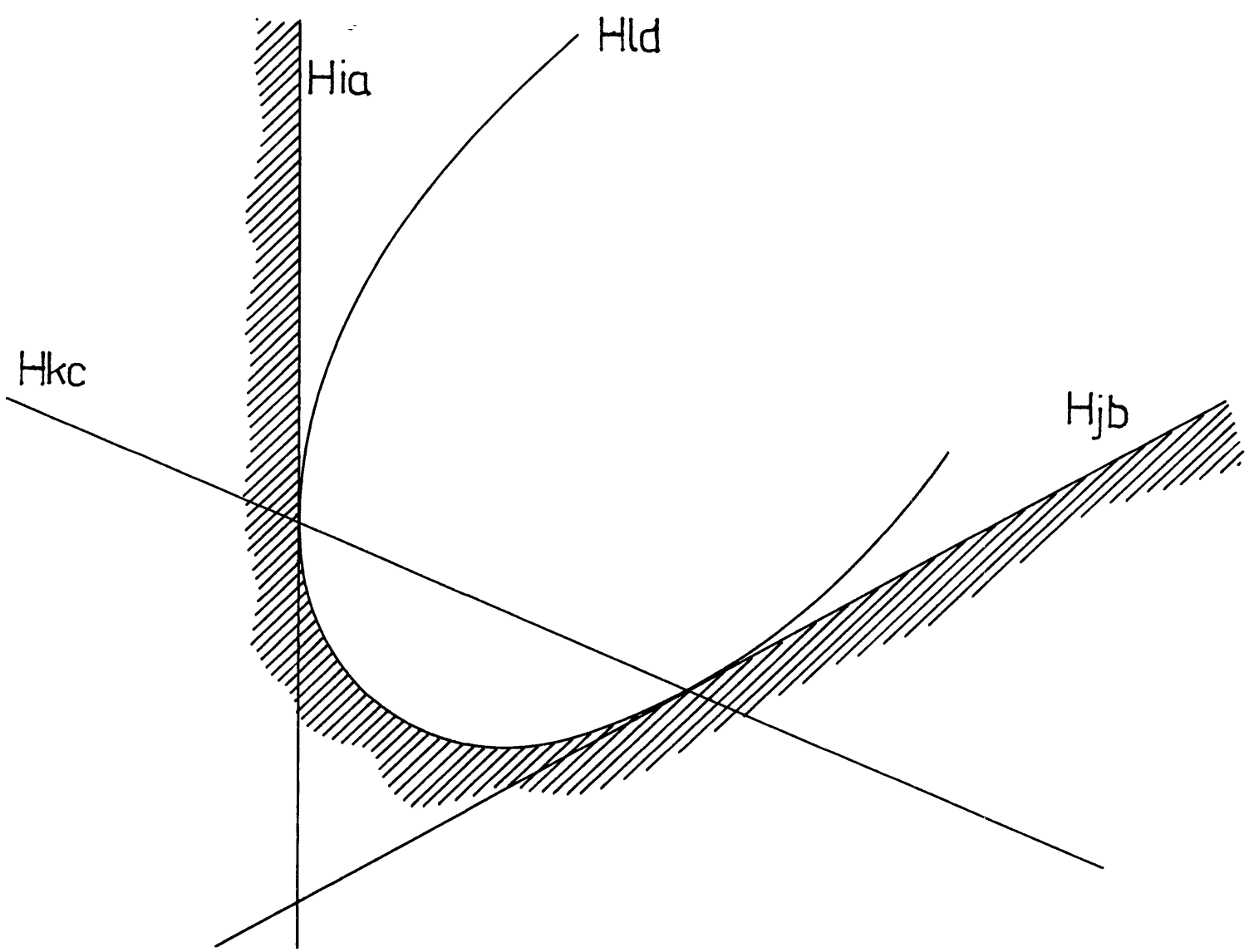


fig.3.16 corner blend

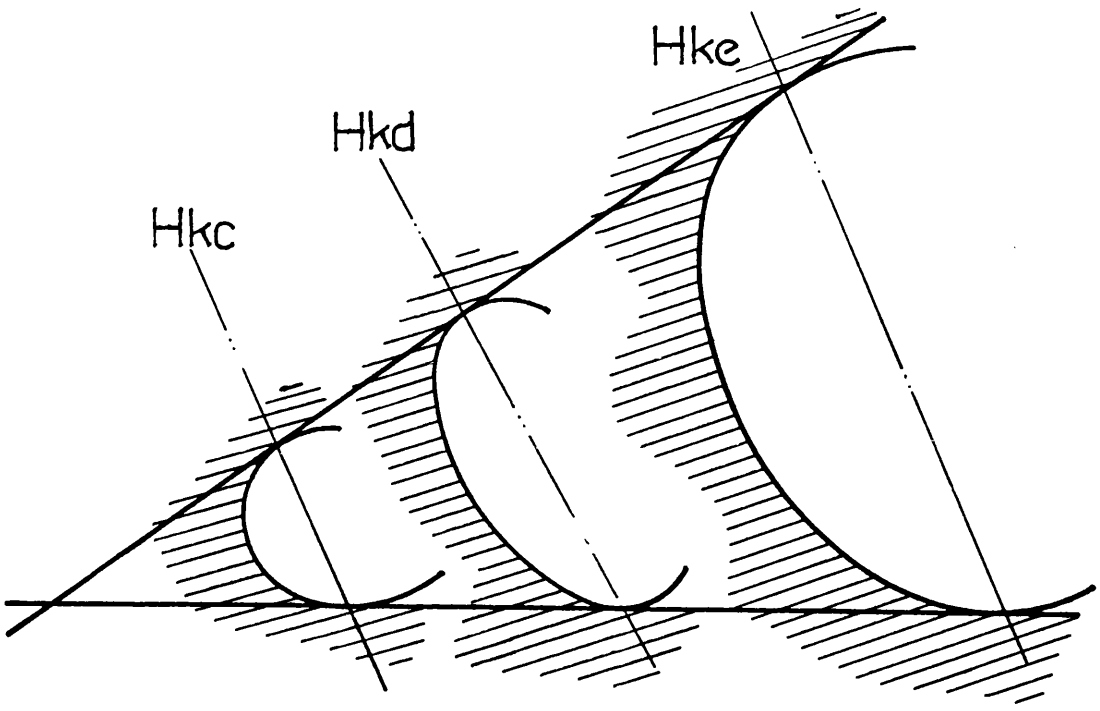


fig.3.17 range control of corner blends

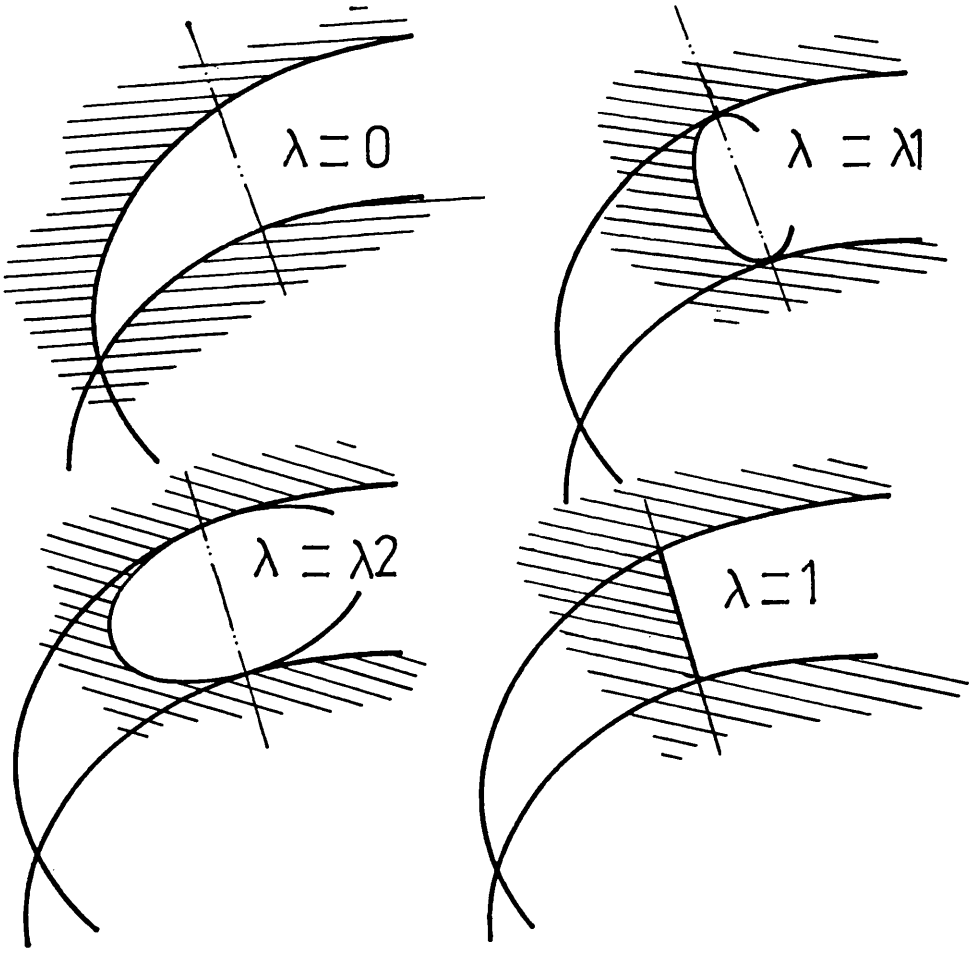


fig.3.18 extent control of coner blends

$$MODEL = H_{ia} \bigcup H_{jb} \bigcup (H_{ld} \bigcap H_{kc}) \tag{3.21}$$

3.4.2 Applications and Discussions of Corner Blend Surfaces

The idea of defining and controlling corner blend surfaces is simple, but in order to use it in a solid modelling system, more investigations and analyses are necessary. In this section, some examples of corner blend surfaces and discussions about their associated problems and other features are given.

3.4.2.1 Choice of Ranging Surfaces

As was described earlier, the range of a blend surface is controlled by the ranging surface. The area of blended surfaces is effectively controlled by the intersection curve between the blended surfaces and the ranging surface. However, the same space curve can be generated by intersecting different surfaces. Though the shape of the blended surfaces are determined by the shape of the object to be modelled, the ranging surface, since it does not appear on the object, can have different shapes, and even different number of ranging surfaces are possible. For example, two curves are needed to define the range of the blend between a cylinder and a plane (Fig.3.19). One curve is needed on the cylinder and the other on the plane. A sphere or a cone can be used for this, but we can also used a plane and a cylinder, as illustrated in Fig.3.20, where the ranging plane is used to cut the blended cylinder to generated a curve on the cylinder and the ranging cylinder is used to cut a curve on the blended plane. The following are the equations for the definition of the corner blend between the plane and the cylinder using the multistage method.

$$H_{2b} = H_{1k}^2 + H_{1m}^2 - R_b^2 \tag{3.22a}$$

$$H_{2r} + H_{1k}^2 = H_{1m}^2 - R_r^2 \tag{3.22b}$$

$$H_{6b} = (1-\lambda) H_{2b} H_{1a} - \lambda (H_{1r} H_{2r})^2 \tag{3.22c}$$

where H_{2b} and H_{1a} are the blended cylinder and plane, H_{2r} and H_{1r} are the ranging cylinder and plane, H_{6b} is the blend surface and λ is the control constant.

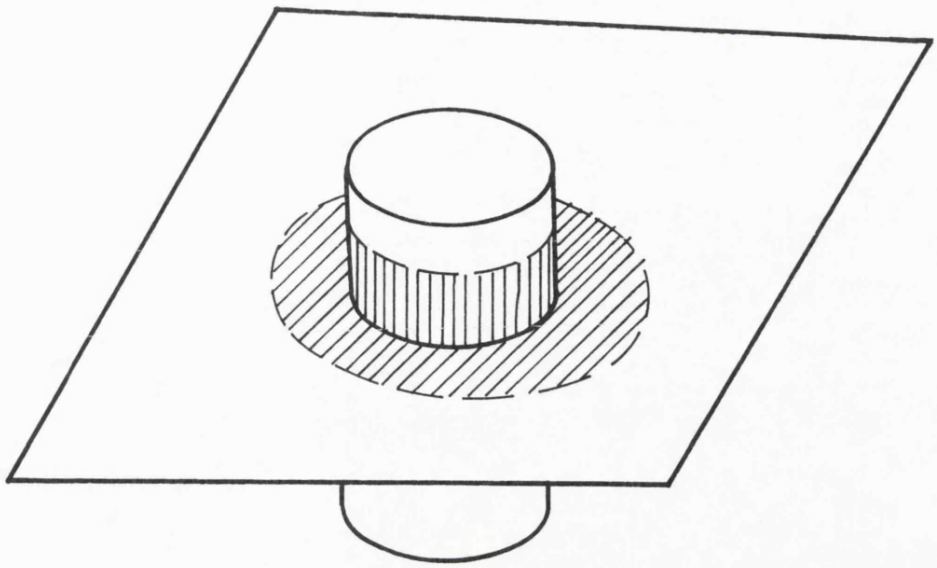


Fig.3.19 Bounding Curves for Blending a Cylinder and a Plane

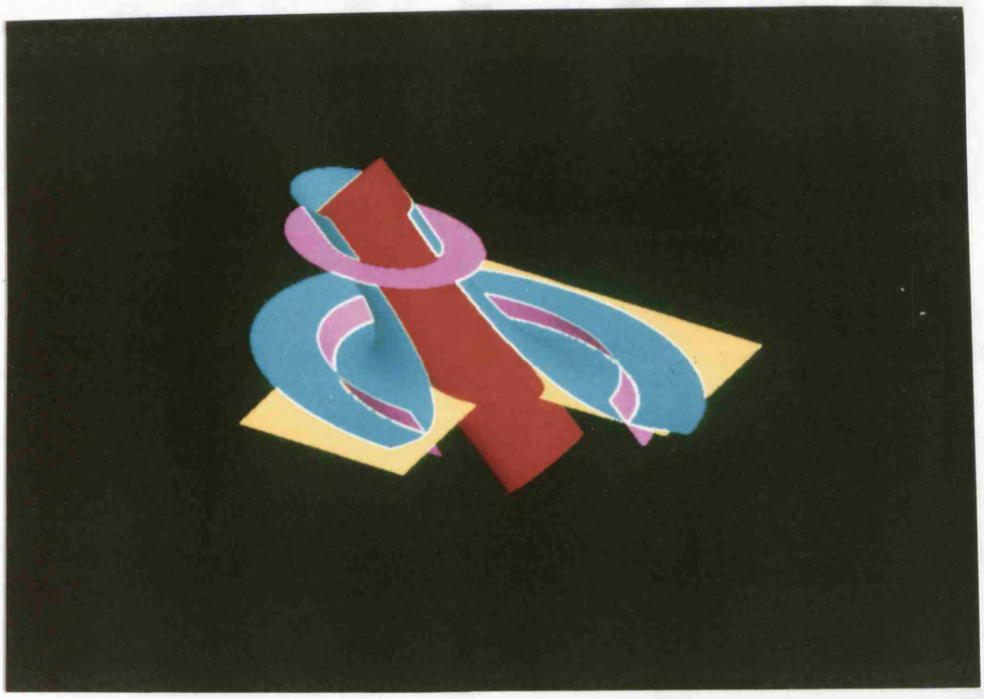


Fig.3.20 A Cylinder and a Plane Used as the Ranging Surfaces
for Blending a Cylinder and a Plane

As there are enormous numbers of different blended surfaces and even the same blended surfaces can combined in many different ways, it is hard to use the same ranging surface, or ranging surfaces, to deal with all situations properly. It is possible to define ranging surfaces in a way that certain features can be automatically controlled. For example, fixing the number of ranging surfaces to be the same as the blended surfaces can enable them to be defined in terms of the blended surfaces. But because of the reasons mentioned above, there are always some situations that cannot be handled by such simplifications.

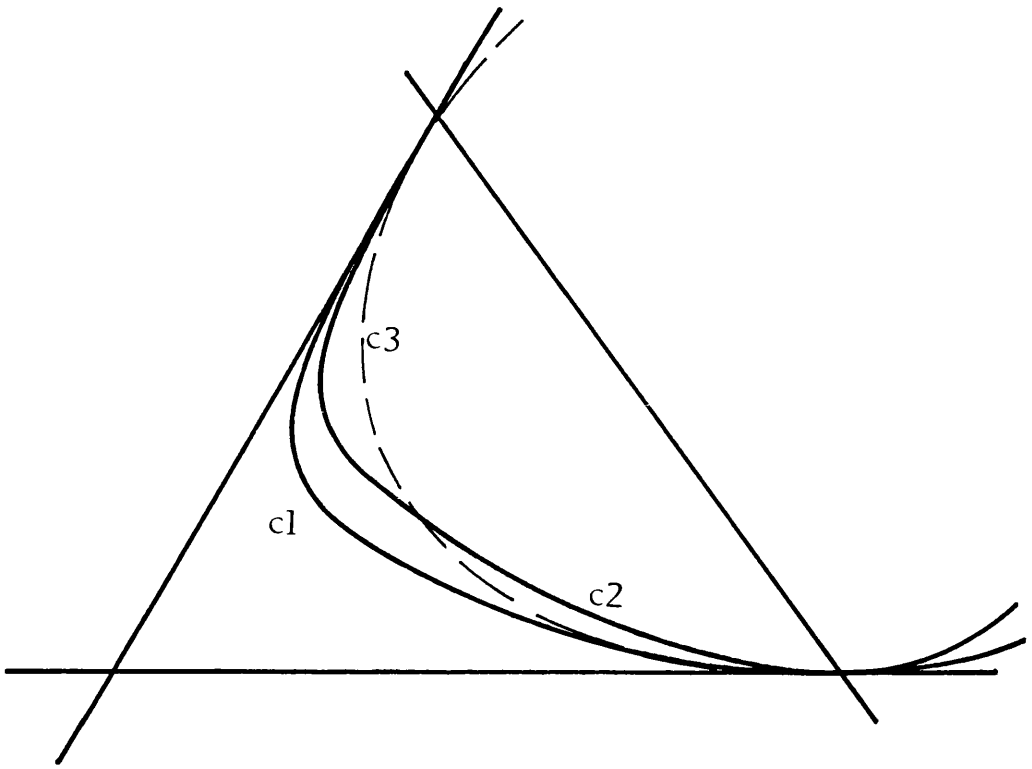
In this thesis, the choice of ranging surface is left open, that is to say, different numbers and shapes of ranging surfaces can be chosen for different situations. This is not as simple to use as other fixed approaches, but it is more flexible and can handle potentially much more different situations and special cases.

3.4.2.2 Extent and Bias Control of Corner Blend Surfaces

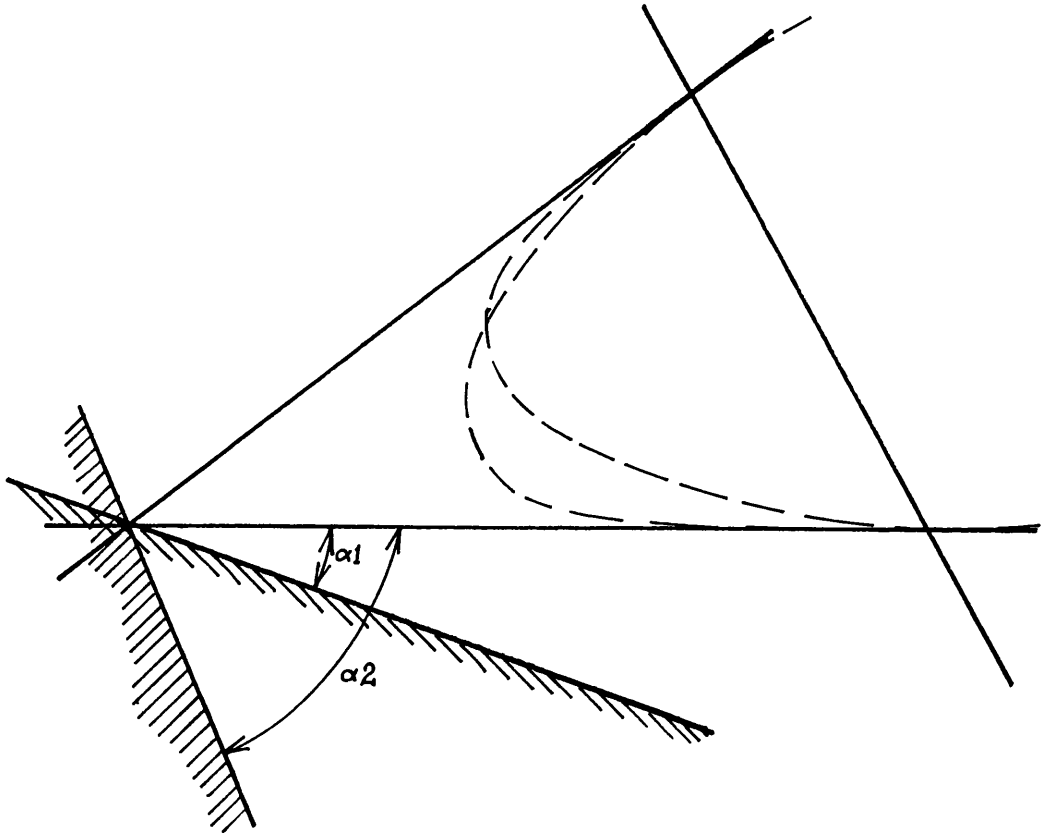
As said earlier, the extent of a blend is controlled by the value of λ. This is satisfactory in many cases where low order surfaces are blended together. However, when high order surfaces are also used, especially when a high order surface is to be blended with a low order surface, this method becomes less convenient to use. The blend is closer to one of the blended surface than to the other no matter what value is given to λ, as shown in Fig.3.21a.

The main reason for this is that the blended surfaces and the ranging surfaces are treated as two groups of surfaces. By changing the value of λ, we can only modify the relative position of the blend between the two groups of surfaces; the relative position of the blend between the blended surfaces can not be modified simply by changing the value of λ.

To solve this problem, more controlling data are needed. In generating models for this thesis, the author used the following method to control the relative position, or *bias*, of the blend surface between the blended surfaces. In addition to the constant λ, a half-space is used as a variable controlling data. A plane or other simple surfaces are usually used for



(a) c1,c2: Blends Controlled by λ c3: The Blend Required



(b) Bias Half-Spaces

Fig.3.21 Bias Control of Corner Blends

this purpose, such a half-space is called the *biasing surface*. The biasing surface is placed near the blended surfaces as shown in Fig.3.21b. By controlling the orientation of the biasing surface (the angle α in Fig.3.21b, for instance), the bias can be effectively controlled.

In engineering, blend surfaces that meet certain requirements are needed, such as those that have constant radius (the result of a rolling ball), constant volume of the blended region, or constant distance between the boundary curves and their opposite blended surfaces. For simple and low order surfaces, these are not too difficult to achieve, but in general, it is hard to meet these requirements precisely. The extent and bias control method give us a more general, more flexible way to control the shape of the blend surfaces. Using this method blend surfaces can be controlled to meet most of the requirements in mechanical engineering. This method can also be used internally by appropriate programs to control the blend in terms of other parameters, such as those mentioned above.

3.4.2.3 Bulge Problem with Corner Blend Surfaces

The basic role of blending is to link two or more surfaces together smoothly with at least slope continuity at the joins of the blend and the blended surfaces. If the slope continuity condition is already met by the blended surfaces themselves, obviously, there is no need to put a blend on them. However, there are situations when two surfaces are partially tangent to each other, such as two intersected cylinders of the same diameter (Fig.3.22). Care should be taken when defining blends between such surfaces, otherwise, an unwanted *bulge* may be produced as a result. A bulge is the extra blend that meets the continuity requirement at the boundary but has too much volume, which cannot be effectively controlled by the constant λ. It is difficult to eliminate such a bulge without eliminating other useful parts of the blend.

As we described earlier, the ranging surface determines the area to be blended. The intersection curves of the blended surfaces and the ranging surfaces are the boundary of the blend, while the blended and ranging surfaces themselves are the lower and upper bound of

the extent of the blend. A bulge occurs when the lower and upper bounds of the extent are not equal over areas where the tangent condition is already met by the blended surfaces.

The bulge problem can be solved by using properly chosen ranging surfaces that are tangent to both of the blended surfaces where they themselves are tangent to each other. When this condition is met, no matter what value λ has, the blend at such places will always by the same, while the blend at other areas can be controlled properly.

Fig.3.22 shows a blend at the joint of two pipes of the same size. Two planes and a cylinder are used as the ranging surfaces with the two planes tangent to both of the pipes at the front and rear sides.

Fig.3.23 shows another example, where a blend is generated at the joint of a cylinder and a sphere, which are tangent to each other at the top of the sphere. The bulge is eliminated by putting the ranging surface (a cylinder) tangent at the same place.

3.4.2.4 Blending Multiple Surfaces

In most applications, two surfaces are blended together. But very often, blends of three or more surfaces are also needed. Such a blend can be generated by increasing the number of blended surfaces in the blend equation (3.20).

For example, the blend at the corner of three planes, as shown in Fig.3.24. The equation of this blend is

$$H_{3a} = (1-\lambda)H_{1a}H_{1b}H_{1c} - \lambda H_{ld} = 0 \qquad (3.23)$$

where H_{1a}, H_{1b} and H_{1c} are the three blended planes and H_{ld} is the ranging surfaces.

Using the same technique, more and different surfaces can be blended together.

3.4.2.5 Blending on Disjoint Shapes

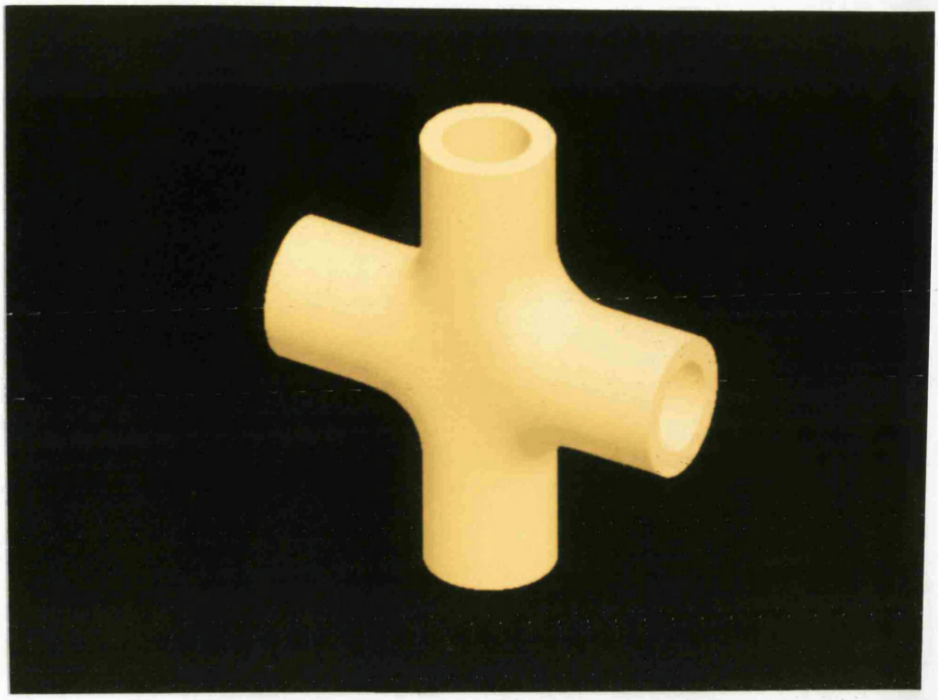


Fig.3.22 A Blend Between Two Cylinders of the Same Diameter

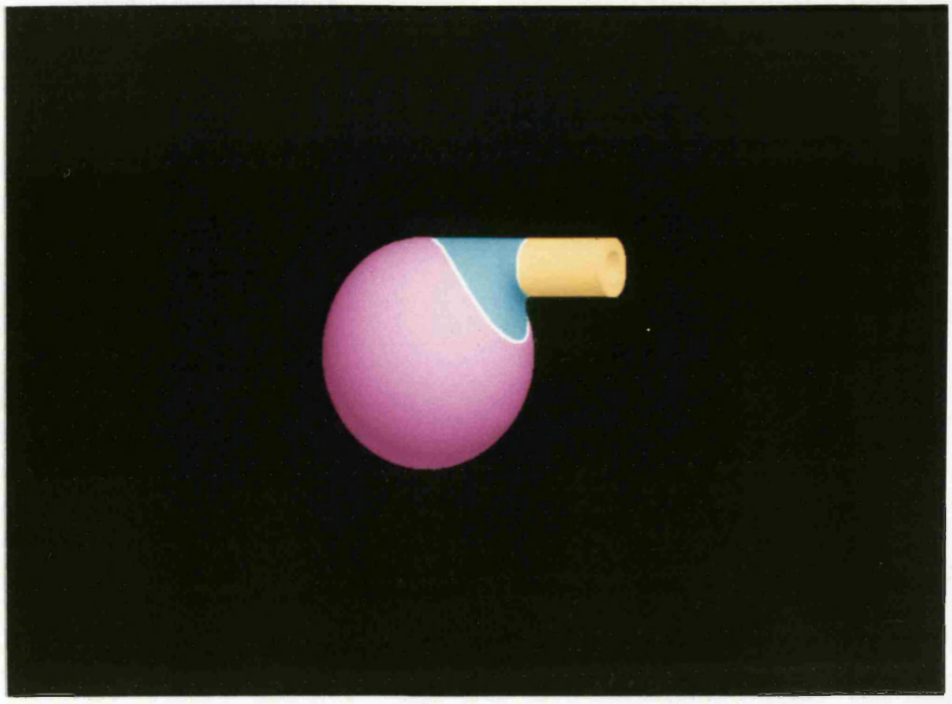


Fig.3.23 A Blend Between A cylinder and a Sphere

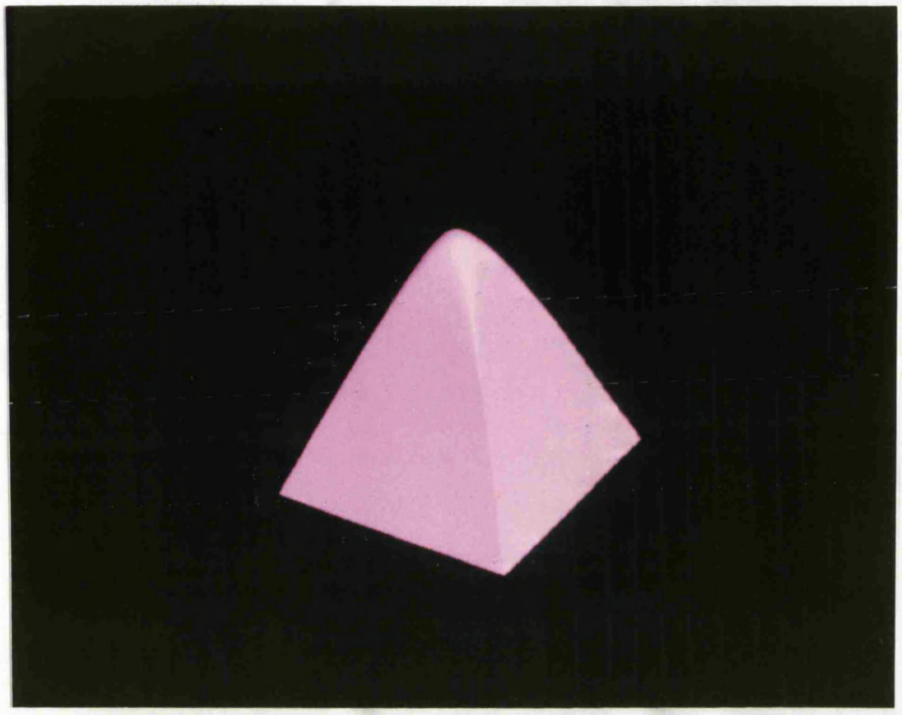


Fig.3.24 Blend at a Corner

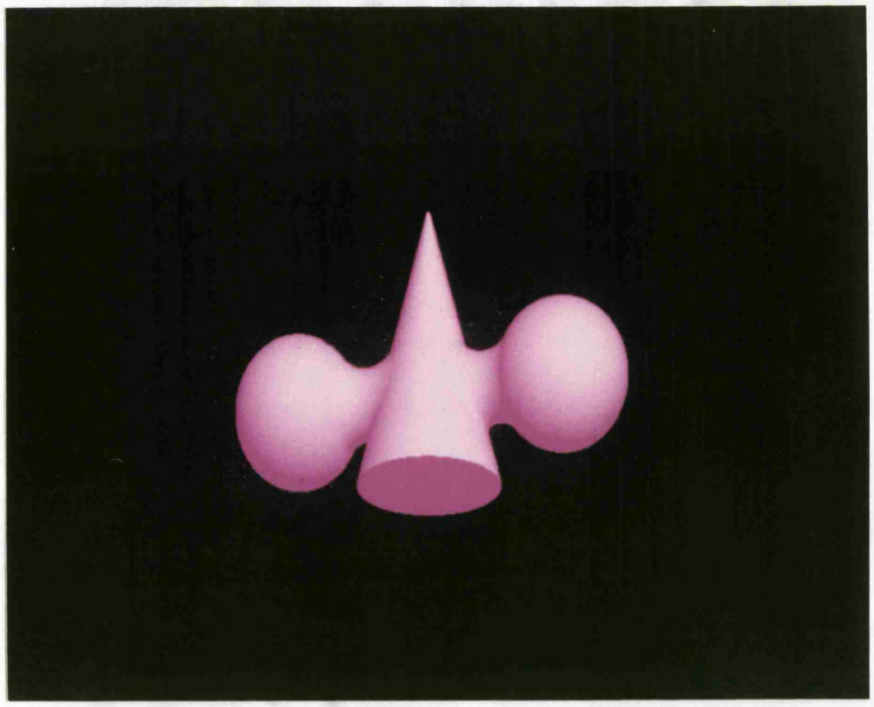


Fig.3.25 Blends Between Disjoint Solids

The corner blend, as its name implies, is usually used at edges or corners of two or more surfaces. However, even when two or more surfaces do not intersect each other, they can still be blended together, as long as the blend boundary is properly defined, and the control data are properly specified.

As an example, a cone is blended together with two spheres, which are disjoint with the cone, as shown in Fig.3.25. Blend surfaces generated in such way also meet the continuity requirement.

3.4.2.6 Blending on Complicated Surfaces and Blends on Blends

The most commonly used blends are those on primitive shapes. But blends on complicated surfaces, and blends on previously generated blend surfaces are also often required in modelling engineering components.

The multistage method of defining implicit polynomials is very easy to use in generating such blends, because, though the surfaces being blended can become more and more complicated, the definition form of the blend remains simple at each definition stage. The elements for this definition can be any half-spaces, as long as they were defined beforehand. For example, the blends at the corners of the hand-wheel in Fig.3.26 were generated by blending on previous blend surfaces. Another example is a hip-joint replacement (Fig.3.27), where blends on complicated surfaces and on previous blend surfaces are defined in constructing this model.

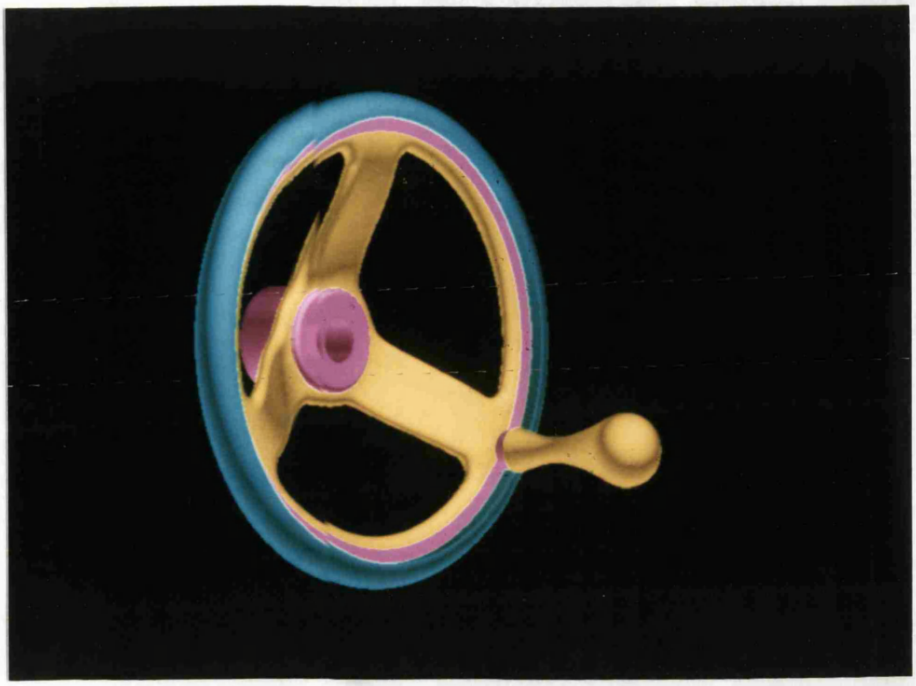


Fig.3.26 A Model of a Hand-Wheel

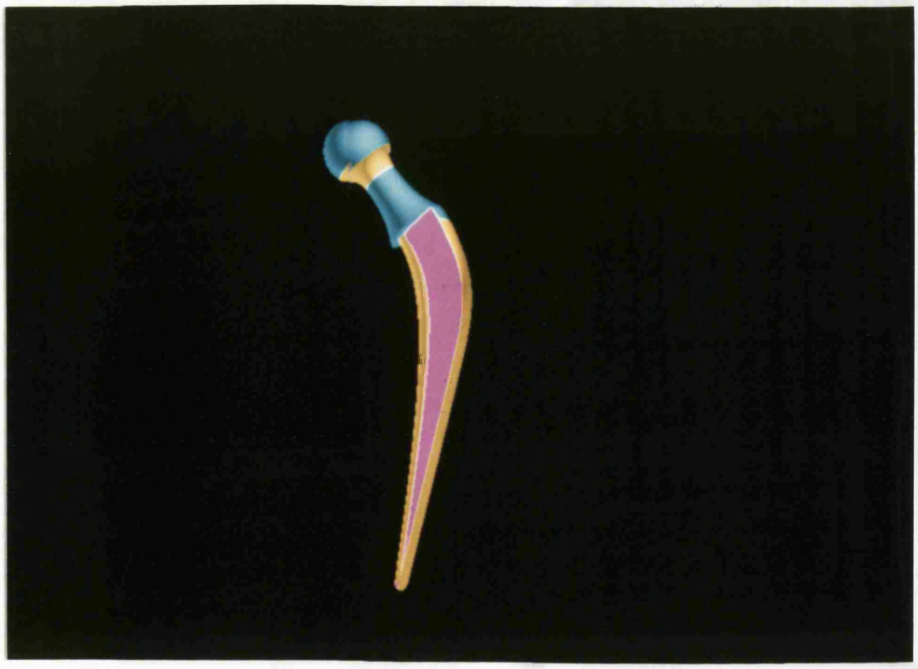


Fig.3.27 A Modlel of a Hip-Joint Replacement

3.5 TECHNIQUES FOR DEFINING GAP BLEND SURFACES

Corner blend surfaces specified by the above techniques have proven to be effective and convenient to use in solid modelling. However, there are other types of blend surfaces (which also appear on many engineering components and other objects) that are difficult to represent by corner blends; for example, the blend between the two concentric cylinders of different diameter at the neck of a wine bottle. In such a situation, the blended surfaces generally lie at different sides of the blend and cannot be blended together by rolling-ball and similar techniques.

Since the blend has to be continuous with both of the blended surfaces, the corner blend is not suitable for this and a different method has to be used. A blend formulation that solves this problem is proposed in this section. Blend surfaces defined using this formulation contain inflexions, and may be called intuitively, *gap blend*. Armed with a blend of this type, a modelling system can represent many different pipes and ducts with smooth transitions from one cross section to another, and also sheet metal work with twists and other complicated deformations.

3.5.1 Basic Idea of Defining Gap Blend Surfaces

To begin with, let us consider blending two lines in two dimensions. Suppose that lines H_{1a} and H_{1b} are to be blended within the area defined by H_{1g} and H_{1h}, which are another two lines, as shown in Fig.3.28. The blend is required to link H_{1a} and H_{1b} together smoothly with tangent points on H_{1a} at the intersection of H_{1a} and H_{1g} and on H_{1b} at the intersection of H_{1b} and H_{1h}. The equation

$$H_{3c} : (1-\lambda)H_{1a}H_{1h}^2 + \lambda H_{1b}H_{1g}^2 = 0 \qquad (3.24)$$

defines a curve H_{3c}, which meets the requirements. λ is a constant within the range [0, 1], as before. Fig.3.29 and Fig.3.30 give some examples of blend curves for two lines under different circumstances.

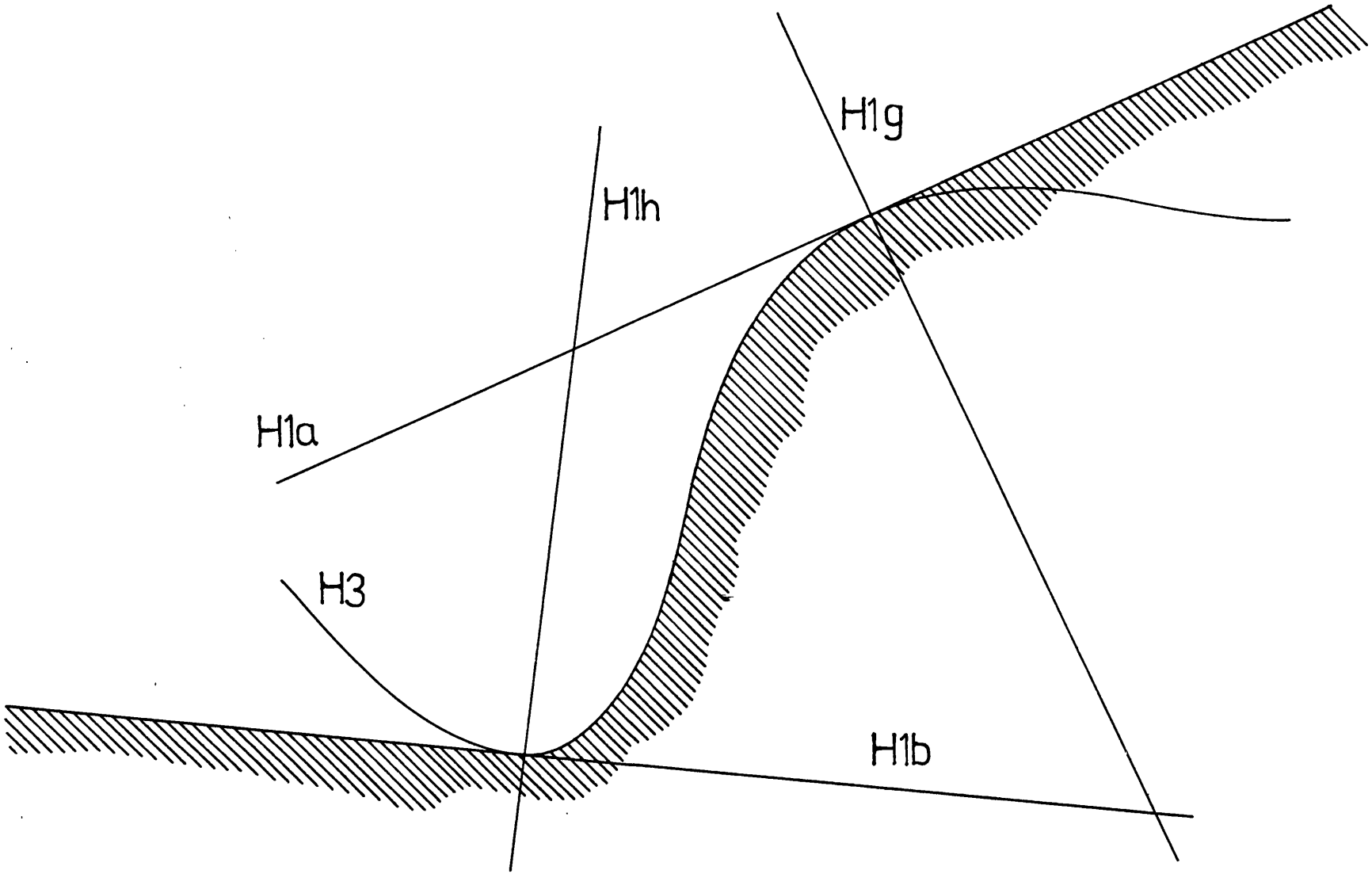


fig.3.28 gap blend on two lines

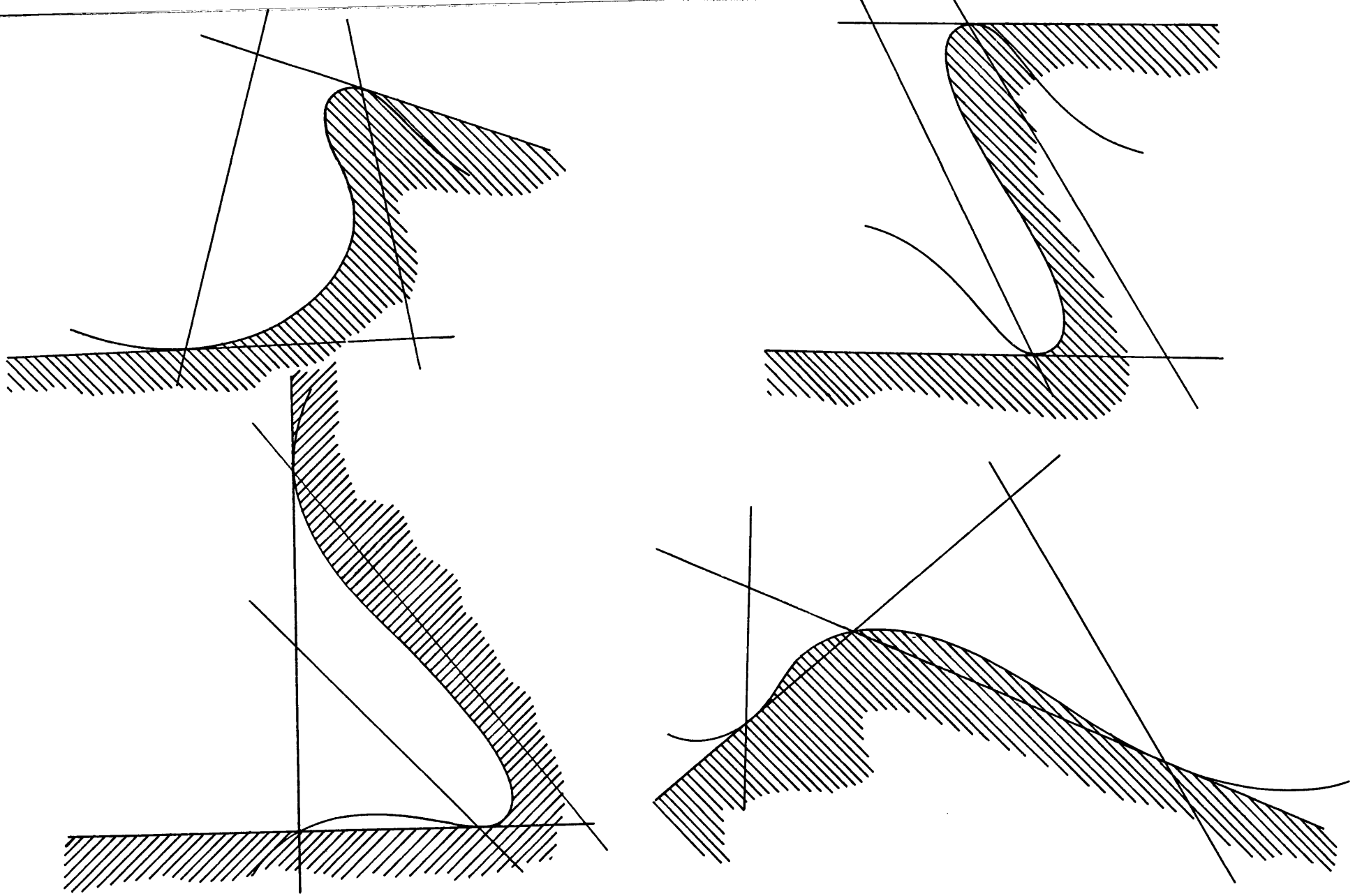


fig.3.29 range control of gap blends

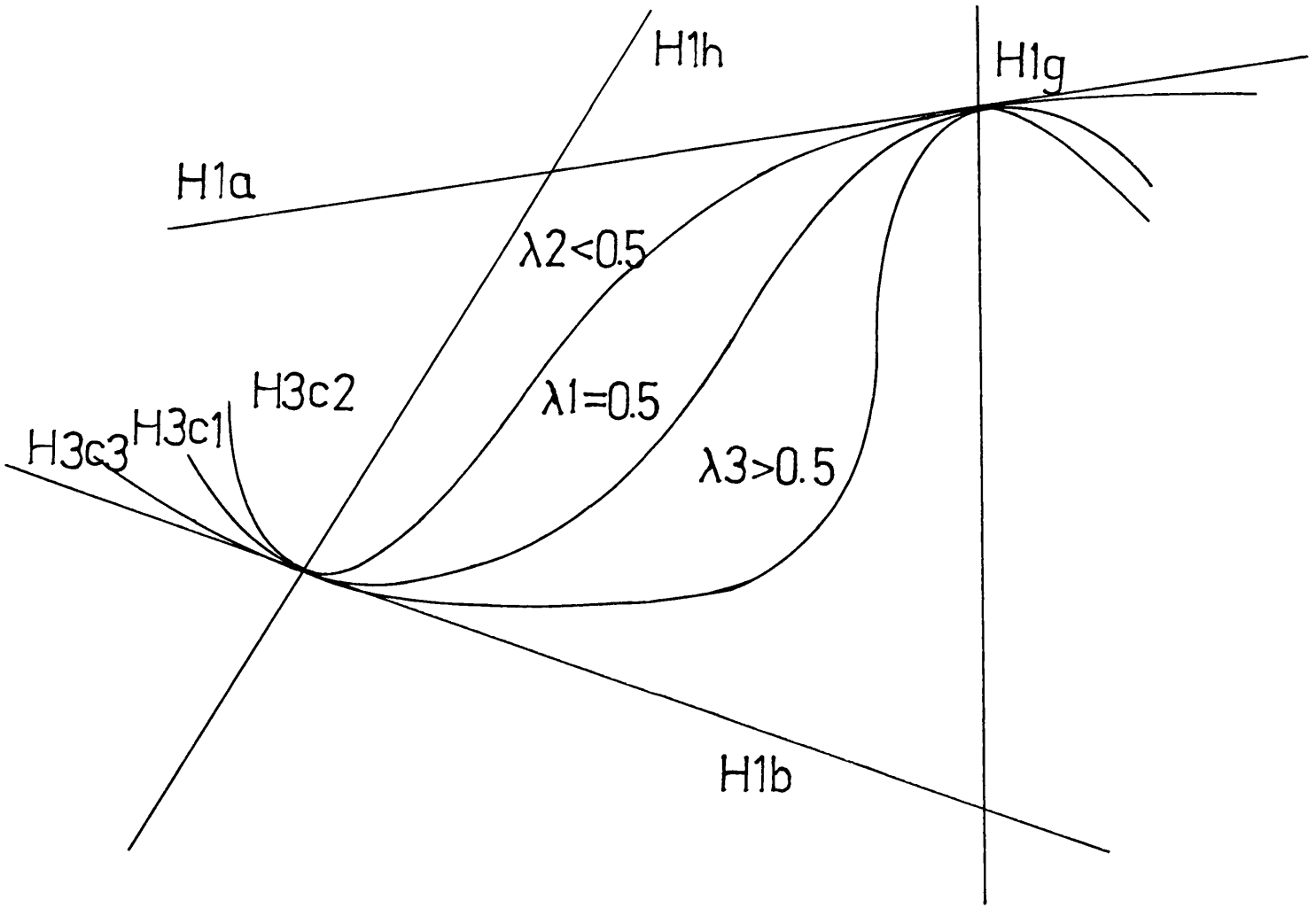


fig.3.30 extent control of gap blends

Apart from the equation form and the shape of blend surfaces, this formulation has similar properties to that of the Liming's equation for a quadratic curve. (The simplest form is a cubic for gap blend.) It meets the continuity requirements and can also be extended in three-dimensional space to blend complicated and high order surfaces.

3.5.2 Applications of Gap Blend Surfaces

The gap blend equation (3.24) can be used directly in three-dimensional space to blend planes by replacing lines with planar half-spaces. The equation is exactly the same as (3.24) if the lines are thought of as planes. In three dimensions, more complicated relationships between planes can be established, and, not only can planes be blended in the same way as blending lines in two dimensions, they can also be twisted. The effect of twisting a plane can be achieved by blending two intersected planes along the intersection line within the area defined by another two ranging planes perpendicular to the intersection curve. The angle to be twisted is controlled by the angle between the two blended planes, the length of the twisted part is controlled by the distance between the two ranging planes. The continuity requirement is automatically satisfied by the equation. Compared with other methods using parametric patches or numerical interpolating techniques, this one is easier to use and control. Shifting the positions of the blended planes and ranging planes in the equation, a blend between two parallel planes can be defined. This generates the effect of curling a plane. Examples of applying gap blend on planes are shown in Fig.3.31.

Curved surfaces can be blended just as easily by substituting their equations for those of the planes. For example, cylindrical shapes of different size and cross-section can be easily blended together smoothly. The blends on the model in Fig.3.32 are some examples.

In practice, the cylindrical surfaces need not have the same axis, and even when the surfaces intersect each other, they can still be blended together without losing the continuity property. A typical example is given in Fig.3.33.

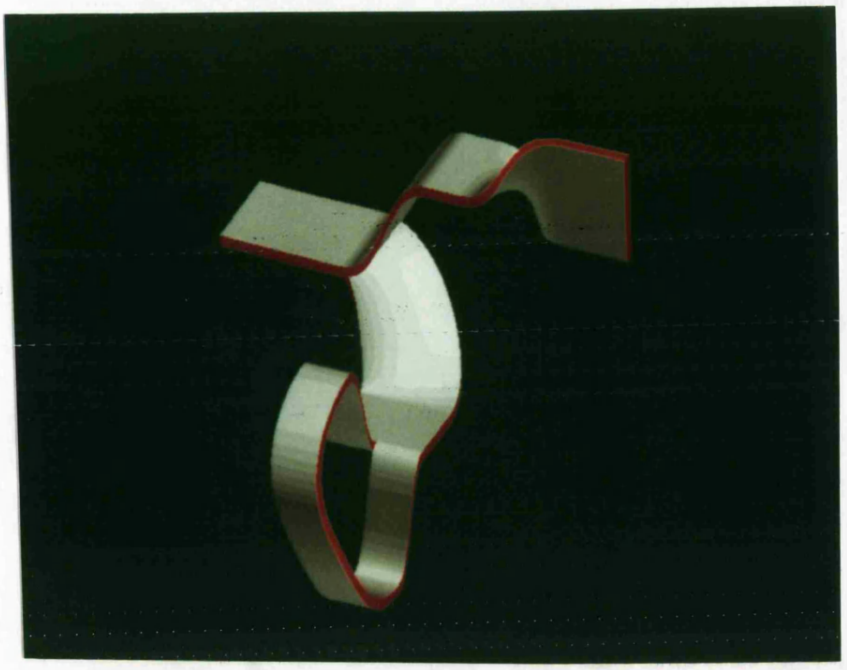


Fig.3.31 Gap Blend Between Planes

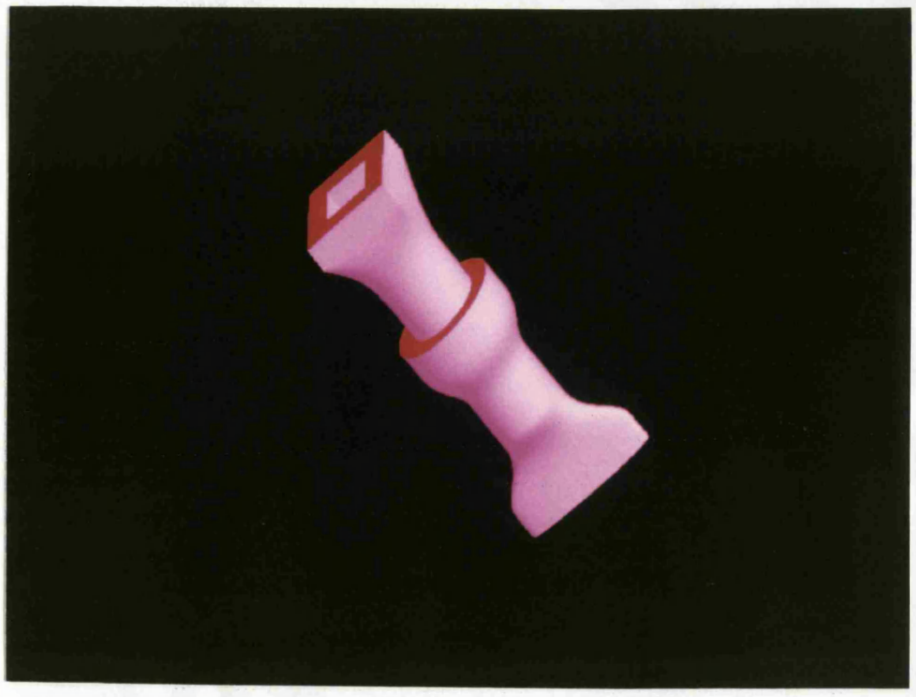


Fig.3.32 Gap Blend Between Pipes of Different Cross-Sections

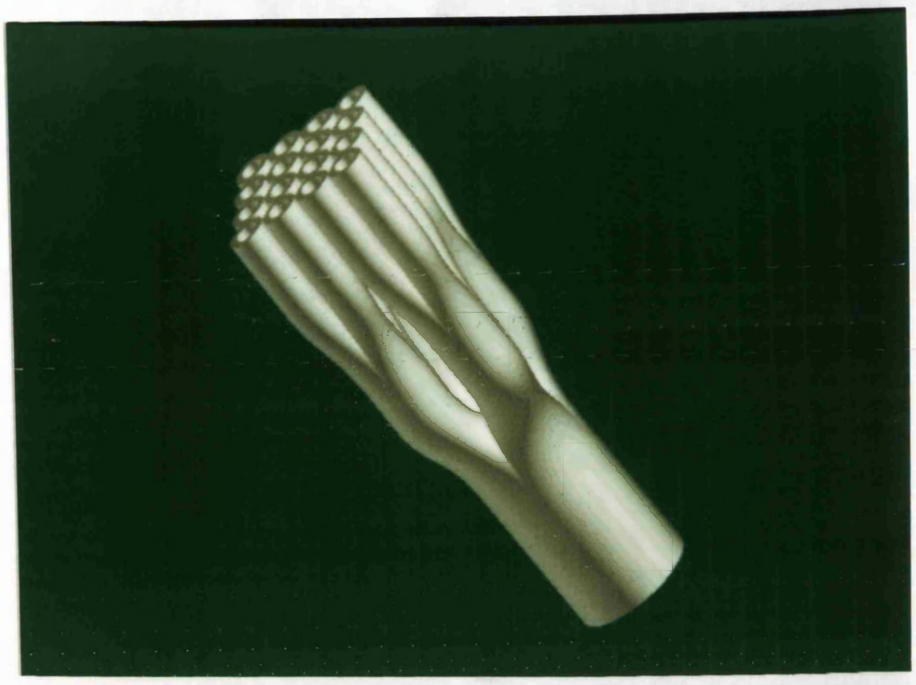


Fig. 3.33 Blends Between Cylinders of Different Diameters and Central Lines

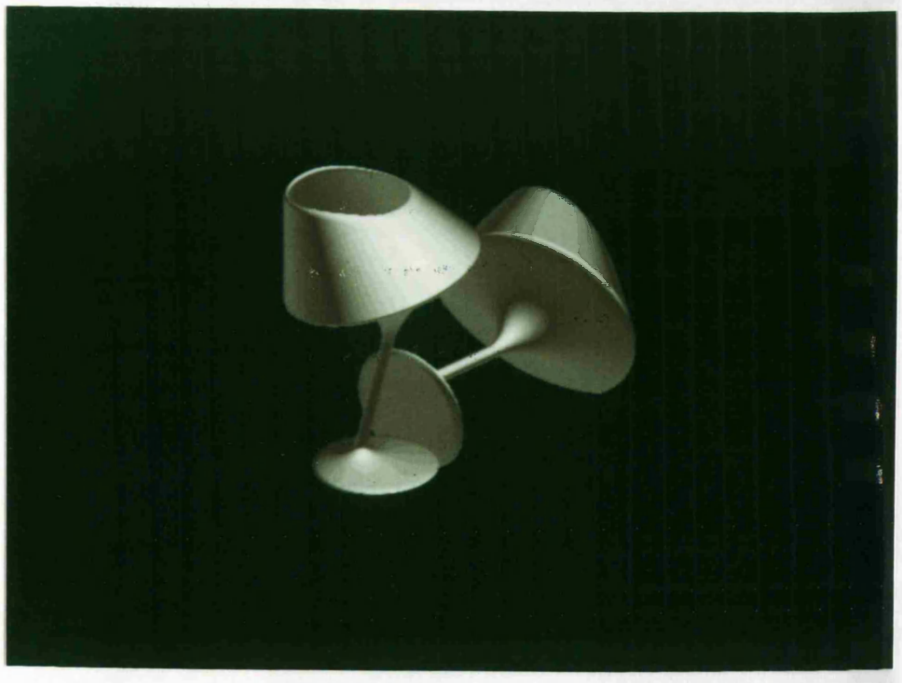


Fig.3.34 Blends Between Cones and Cylinders

Not only cylindrical surfaces can be blended in this way, other curved surfaces can also be used. For example, a cylinder in an arbitrary orientation inside a cone can be blended with the outside surface of the cone smoothly, as shown on the cups in Fig.3.34.

A blend between a cylinder and a plane can be defined as a corner blend when they intersect each other. But when they become parallel, or nearly parallel, corner blends becomes difficult or even impossible to use and the gap blend can be used instead. This kind of blending involves spreading a curved surface into a planar surface, or into another surface which curves away from the first. As illustrated in Fig.3.35, gap blends are very suitable for defining such blend surfaces.

3.5.3 Discussion and Generalisation of Gap Blend

3.5.3.1 Continuity

Continuity between blends and blended surfaces is one of the basic and most important aspect of blending. Equation (3.24) defines blend surfaces that have slope continuity. These are satisfactory for many applications in engineering, but there are other objects which require different degrees of continuity. For example, straight joints with only positional continuity, and curvature continuity where aesthetic smoothness or functional requirements (such as smoothness for acceleration and deceleration for dynamic fluid devices or cams) demand a higher order continuity than just slope continuity.

The continuity between a blend and blended surfaces is determined by the order of the ranging surface. If the equation

$$H_{qc} : (1-\lambda)H_{ia}H_{nh}^{k} + \lambda H_{jb}H_{mg}^{k} = 0 \qquad (3.25)$$

is used, blends of different degrees of continuity with the blended surfaces can be generated. Here k is normally 1, 2 or 3, for linear, slope or curvature continuity respectively. For example, in Fig.3.36, k is 1 for Fig.36.a and Fig.36.b and linear continuity is achieved. When the blended and the ranging surfaces are linear surfaces themselves, the cross-section

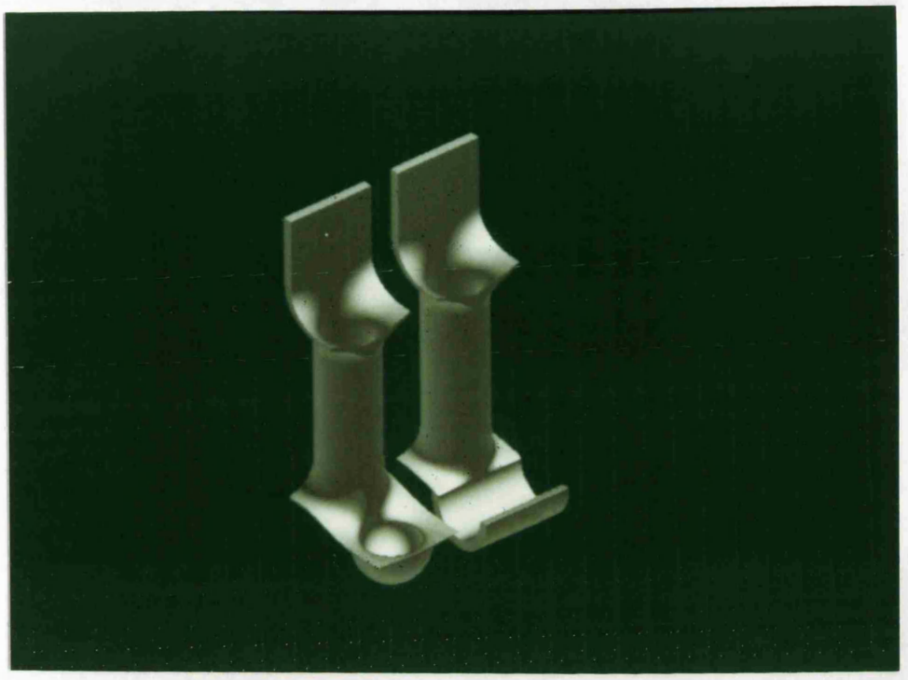


Fig.3.35 Blend Between Surfaces Curving Away From Each Other

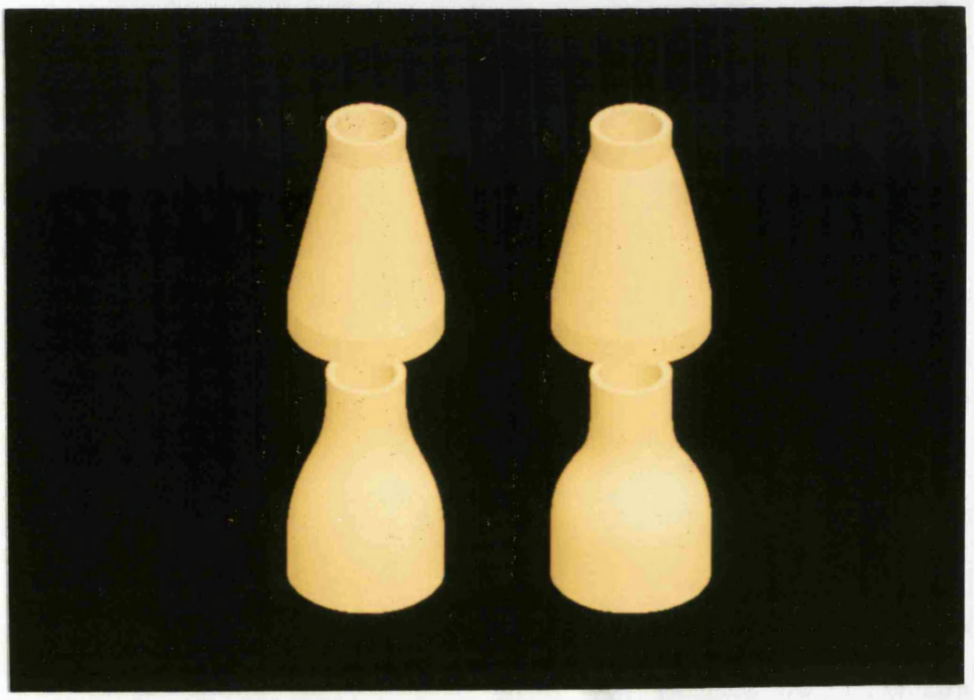


Fig.3.36 Continuity Control of Gap Blends

a	b
c	d

of the blend is linear, too, as in Fig.36.a; otherwise, the cross-section of the blend is curved, as in Fig.36.b. In Fig.36.c, k is 2 and slope continuity is obtained. This is used for the majority of models in most applications. When k is 3, curvature continuity can be achieved, as shown in Fig.36.d. One important feature of the author's method is that the continuity of the blend with the blended surfaces is independent of the absolute order of the blended surfaces and the ranging surfaces, and is controlled effectively by the *relative order* of the ranging surface, as given in equation 3. This makes the method very easy to use in practice.

3.5.3.2 Controllability

Another basic requirement for blending is that for control over the blend surfaces. The range of the blend is determined by the relative position and orientation of the ranging surfaces. The gap blend can be put wherever wanted, as shown in Fig.3.29, in a similar way to the corner blend, by the user's properly choosing the shape, position and orientation of the ranging surfaces. This shows an advantage over other methods where the bounding curves and continuity conditions have to be explicitly defined.

The extent of the blend within the blending area is effectively controlled by the value of λ, as in Fig.3.30. When the difference between the order of blended surfaces and the ranging surfaces becomes big, it becomes more difficult to choose the right value of λ, but with interactive feedback, the value of λ can be decided easily by judging the appearance of the blend cross-section or by requiring the blend to pass through a given point. Plane sections can be used to give a quick view of a model, part of a model, or even a single curve of a polynomial to check the correctness of a blend and to decide if modification is needed [Chapter 4]. This is also a convenient way to determine the value of λ and other control data to obtain the required blend shape.

Another factor that ought to be controlled in some circumstances is the *bias* of the blend between blended surfaces. This can be done by using biasing planes, lines, points, or

some other surfaces. For example, the blend between two cylinders in Fig.3.37 is biased towards the right hand side by placing a biasing point (a sphere with zero diameter can be used here) where the change is required. This has a global effect on the entire shape of the blend, but the effect is predictable and controllable, and is also convenient to use.

When biasing is required, the blend equation is extended:

$$H_{qc} : (1-\lambda) r H_{ia} H_{nh}^{k} + \lambda s H_{jb} H_{mg}^{k} = 0 \qquad (3.26)$$

where r and s are the biasing planes, lines, points, other surfaces, or even constants. In most cases, they are set to 1 or other constant values. One or both of them are set as variable entities only when it is necessary. When they are both set to the same constants, they can be used to bring down the magnitude of the potential values of implicit polynomials to avoid numerical problems during their evaluation. This does not change the shape and continuity property of blend surfaces, but effectively solves problems such as floating point overflow, which are common in dealing with implicit polynomials, especially high order ones. Implicit polynomials with a maximum order of 16 were used in constructing the examples for this thesis. Surfaces of higher order can also be used if that is necessary. Generally speaking, there is no restriction on the order of implicit polynomials for defining blend surfaces, as long as numerical problems can be solved, and the means of control over the shape of blend surfaces are appropriate.

3.5.3.3 Multiple Surface Blending

So far, this discussion has concentrated on blending two surfaces together. But in fact, this method can be applied to blending multiple surfaces together. If A and B are used for blended surfaces, G and H are used for ranging surfaces, the gap blend equation can be further generalised to define blend C as

$$\begin{aligned} C : {} & (1-\lambda) r A_1 A_2 \cdots A_m (H_1 H_2 \cdots H_i)^k \\ & + \lambda s B_1 B_2 \cdots B_n (G_1 G_2 \cdots B_j)^k = 0 \end{aligned} \qquad (3.27)$$

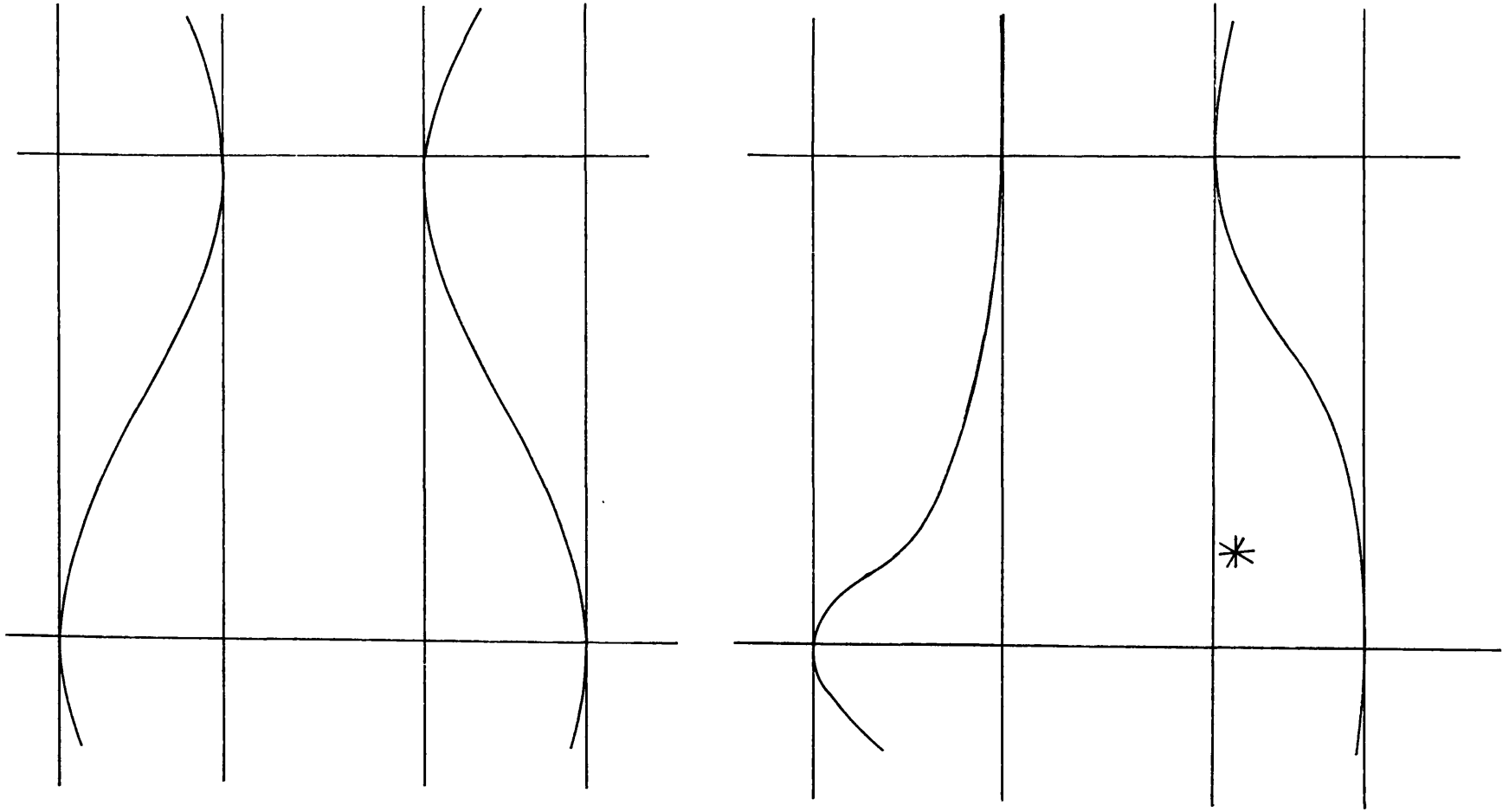


fig.3.37 bias control of gap blends

where m and n are the number of surfaces in two groups of blended surfaces, i and j are the number of surfaces in the two groups of ranging surfaces respectively, and k is the value needed to achieve the desired degree of continuity between the blending and the blended surfaces, as discussed above. For example, 4 cylinders and 5 holes in Fig.3.38 are blended together with two planes as the ranging surfaces, as in Fig.3.39. In Fig.3.40 the surface is generated by blending the two ranging planes together with the 9 cylinders as ranging surfaces. Here k is 2 and slope continuity is obtained.

It should be made clear that, though the generalised form of gap blend surfaces is given as (3.27) to show its capability and versatility, in very few cases is it used fully. For most of the author's applications two surfaces are blended together (including blends on blends, of course), and the ranging surfaces are usually simple ones like planes or quadrics, and only slope continuity is required, hence k is usually 2.

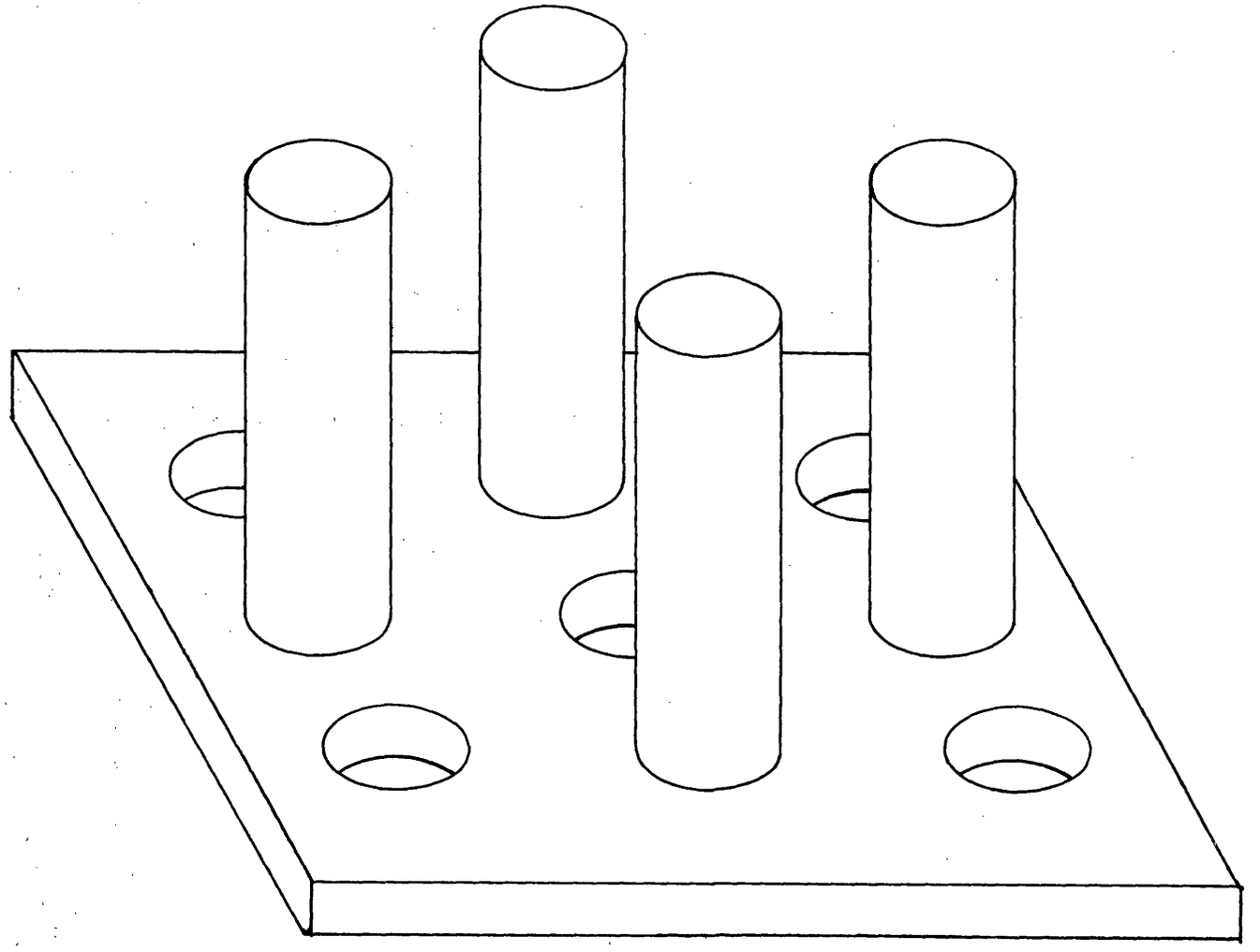


fig.3.38 four cylinders and five holes to be blended

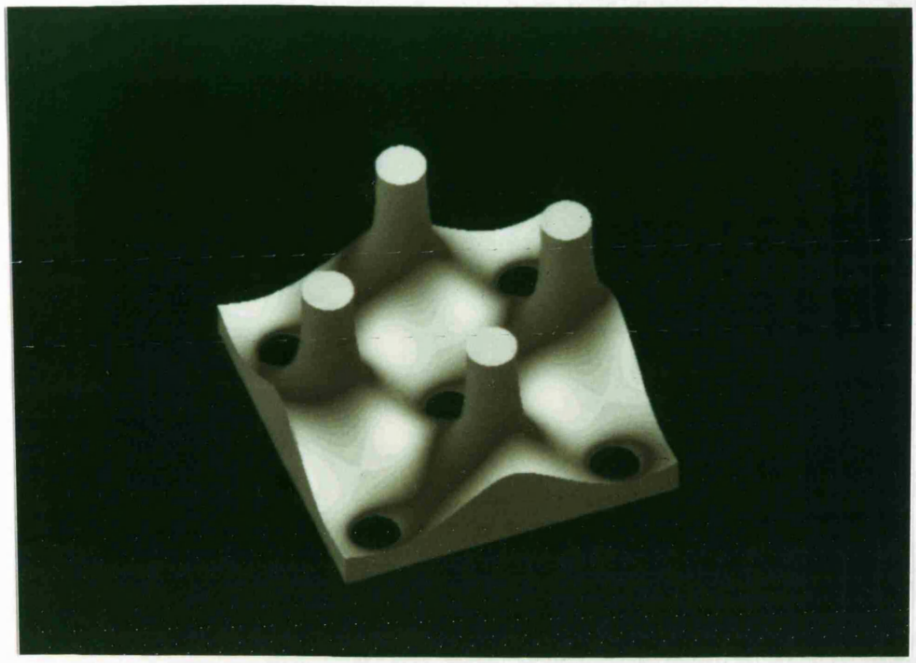


Fig.3.39 Blend Between Four Cylinders and Five Holes

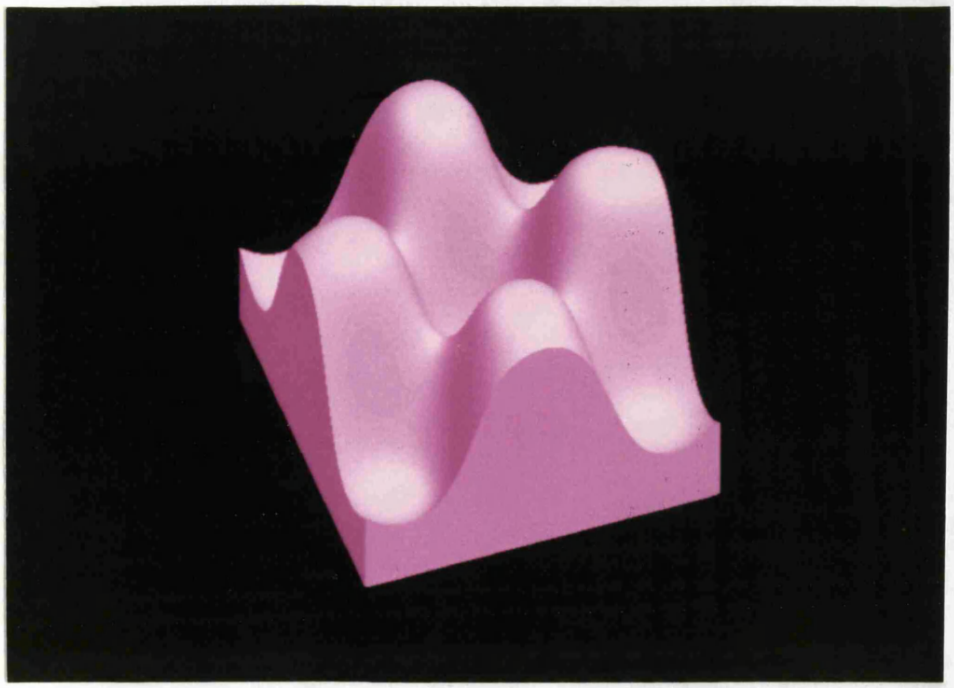


Fig.3.40 Surface Generated by Blending Two Planes with
Four Cylinders and Five Holes as the Ranging Surfaces

3.6 APPLICATIONS OF COMPLICATED AND BLEND SURFACES IN SOLID MODELLING

In this chapter, we discussed techniques for defining various implicit polynomials, which include simple ones like planes and quadrics, complicated high order surfaces, and corner and gap blend surfaces. It it the multistage method that makes it simple to define complicated high order surfaces. Using techniques described in this chapter, we can define many different kinds of surfaces that cover most surfaces needed in mechanical engineering.

Because of the fundamental consistency of various surface types defined by this method, a solid modelling system can be more powerful and reliable in modelling various objects. The corner blend and gap blend surfaces can represent most of the blend surfaces in mechanical engineering. Using them both, together with other complicated surfaces, far more complicated objects can be represented accurately.

In Fig.3.41, the corner and gap blend techniques are used to defined the surface of the Klein bottle. Fig.3.42 and Fig.3.43 are another two examples of engineering parts. Compared with other models shown in this thesis, they are obviously geometrically more complicated. Without the combinatorial use of various simple and complicated surfaces and blends between them, it would be difficult to represent such objects.

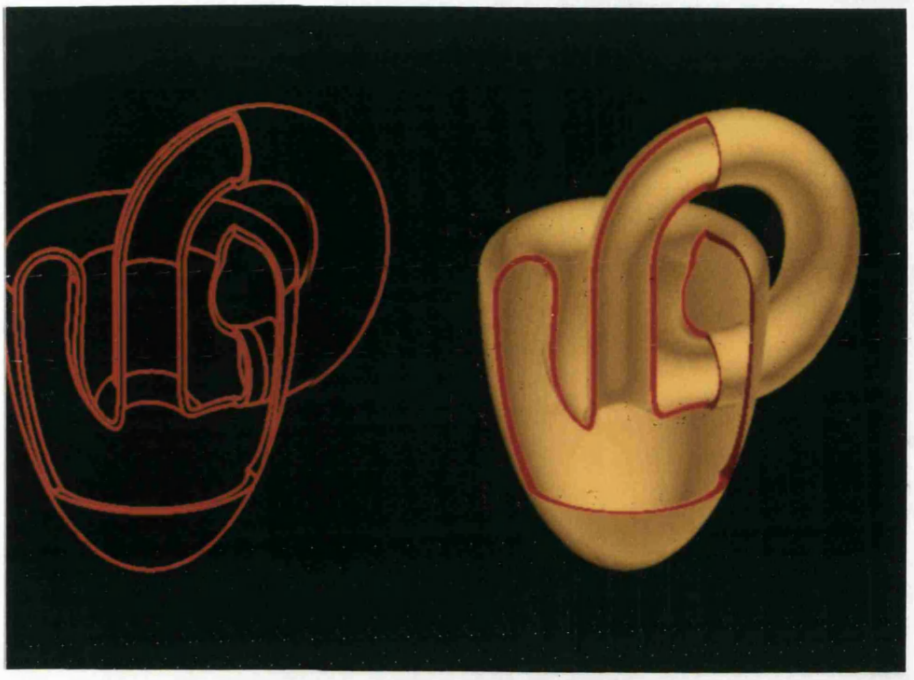


Fig.3.41 Wire-Frame and Shaded Picture of a Klein Bottle

Fig.3.42 A Component with Both Corner and Gap Blends

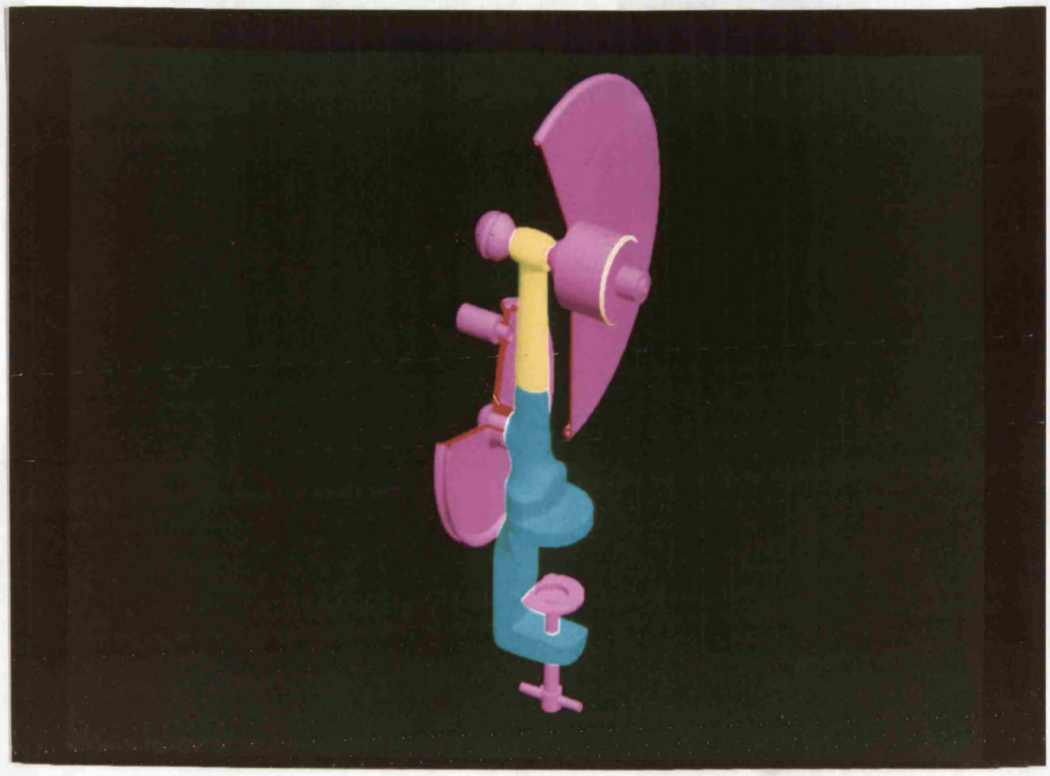


Fig.3.43 A Model of a Tape Winder Consisting of
Different Primitive Solids, and Corner and Gap Blends

CHAPTER 4

INPUT TECHNIQUES

In order to model a component using a computer, its shape and size, as well as its position and orientation in space have to be decided. If an assembly of a group of components are to be modeled, their relative positions must also be specified. To do this, the definition of half-spaces and the description of their combinations must be expressed in a form that the computer understands. However, a form that a human can understand easily may not be suitable for a computer and vice versa. Therefore, a conversion is needed between a user description of the model and another form that the computer understands and accepts easily.

4.1 METHODS OF INPUT

The description of the model from the user, together with instructions to handle it, is called the user input. User input can take different forms, depending on the modes of the modeller's operation and its characteristics.

The user input can be entered into a computer (that is, accepted by a program) through different media. For example, a keyboard, tablet, light pen, mouse, joystick, thumb wheel, tracker ball, text file or a combination of these can be used for this purpose (Attempts have also been made to transfer two-dimensional engineering drawings into a computer automatically to construct a model using sophisticated hardware and software [52]).

These devices and files are used mainly in two modes: interactive or batch. Any method that requires man-machine communication during the entering of the description of the model into the computer is interactive. In this mode, certain graphic devices are usually needed to help in carrying out the conversion between the digital information form for the modeller and the intuitive graphic form for the user. This makes the user feel the computer to be more friendly when passing instructions to it or asking for information from it.

The other type of input (batch mode), on the other hand, needs no man-machine dialogue during the construction of the model. Therefore, no graphic devices are necessary at the input stage. In this mode, the complete description of the model is contained in a text file, which is called the input file. The solid modelling system presented in this thesis uses batch input mode. During the input operation, the computer reads in the description of the model sequentially, line by line, from the input file and then converts this into a more compact, better structured form, ready for the modeller to process.

People usually do not like text input for geometric description and operation, but a text input file, apart from its conciseness and logicality, has some good points, some of which are briefly listed here.

a. A description of the shape of an engineering component, unlike that for art work, is not just a temporary thing, but also serves as a document. It describes the component completely and concisely in a form that suits the user and can be used as a communication medium between engineers.

b. Some complicated engineering components are difficult to describe correctly at the first attempt, or even after a few attempts. Errors and mistakes happen very often. A text file, since it remains unchanged after input, enables the user to detect and locate errors or mistakes easily and conveniently. To correct them, he does not have to go through all the steps again. He needs only to modify those detected parts. This saves a great deal of time and effort.

c. A new product may be developed from an old one with little change in its shape or other data. A text file of an old component can be used in the designing of a new product. This contributes to the reduction of the cycle time of product evolution.

d. Sometimes, facilities for interactive input are limited. For example, each interactive input operation needs a terminal, a monitor and some devices such as a tablet. It is not economic and sometimes difficult to allow each user to have such a work-station. A text file can be written at an ordinary terminal, which is much cheaper. If any mistake occurs, the user can print out the text file and do the corrections away from the computer. This also means that the computer is not necessary when the user want to construct a new model, he can do this by hand using a pencil and paper.

4.2 SOME BASIC REQUIREMENTS FOR AN INPUT LANGUAGE

A text file is written in a form convenient for the user and subsequently converted into a form that the modeller accepts and uses. This conversion is done by a compiler like program which is called an input language processor. Just as a FORTRAN or PASCAL compiler transforms a source code file into machine code, an input language processor converts the text description of a model into another form, which is then used as the actual input to the solid modeller.

An input language can take different forms, depending on the features of the solid modeller and the user's preference. But some requirements are basic to all input languages.

a. It should provide means to define all, or even more than, the possible geometric entities that the solid modeller can accept and manipulate. This enables the user to make full use of the modeller and makes further extension of the domain of the modeller possible without changing the input language.

b. It should provide the means to transform defined shape elements, such as a half-space or a primitive, in space. The shape and size of a half-space are geometrically defined, but in order to match it with a model surface or part of the model surface, it needs to be translated, rotated and/or scaled. An input language with transformation facilities makes the description of the model simpler and is more convenient for the user.

c. It should provide a reliable data structure for holding shape elements together and relating them to each other properly in order to form the model. In most cases, set-theoretic operations are employed to meet this requirement. They are used in the same way as those mathematical operators in an input language, but are evaluated differently in subsequent operations by the solid modeller.

d. It should also be able to accommodate graphic information for subsequent picture generation. Information, such as colour, shading and viewing parameters are all necessary for the generation of pictures of the model. They can be given elsewhere, but it is more convenient to put them together with the geometric information about the model in the input language.

4.3 EXAMPLES OF SOME INPUT LANGUAGES

Different input languages have their own syntax rules and symbolic systems. A simpler input language is easier to learn to use, but less convenient for describing complicated models; a more sophisticated one is more powerful for describing complicated models and carries out difficult operations, but may be more difficult to grasp. Examples of three input languages used in the author's work are given here.

Example 1

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Model of a cone blended with two spheres.
;
; commonly used parameters and half-planes.
;
1 = 1
2 = 2
HX: 1.,0.,0.,0.,
HY: 0.,1.,0.,0.,
HZ: 0.,0.,1.,0.,
;
; (1). Construction of the cone.
;
*COLOUR = 2
HY101: 0.,1.,0.,-60.,
HY102: 0.,-1.,0.,0.,
CNS101 = 0.3
CON101 = HX^2 + HY^2 - (HY101*CNS101)^2
MDL1 = HY101.INT.HY102.INT.CON101
;
; (2). The two spheres.
;
*COLOUR = 6
HX201: -1.,0.,0.,-35.,
HX202: 1.,0.,0.,-35.,
HY201: 0.,1.,0.,-20
RAD201 = 16
SPH201 = HX201^2 + HY201^2 + HZ^2 - RAD201^2
SPH202 = HX202^2 + HY201^2 + HZ^2 - RAD201^2
MDL2 = SPH201.UNI.SPH202
;
; (3).The ranging surface: a cylinder
;
RAD301 = 10
CYL301 = HY201^2 + HZ^2 - RAD301^2
;
```

```
; (4). The blend between the cone and the two spheres.
;
CNS401 = 0.9
BLD401 = (1-CNS401)*CON101*SPH201 - CNS401*CYL301^2
BLD402 = (1-CNS401)*CON101*SPH202 - CNS401*CYL302^2
HX401: 1.,0.,0.,0.,
HX402: -1.,0.,0.,0.,
HX403: 1.,0.,0.,-35.,
HX404: -1.,0.,0.,-35.,
MDL401 = BLD401.INT.CYL301.INT.HX401.INT.HX404
MDL402 = BLD402.INT.CYL301.INT.HX402.INT.HX403
MDL4 = MDL401.UNI.MDL402
;
; (5). The final model.
;
MDL5 = MDL1.UNI.MDL2.UNI.MDL4
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

This example describes the geometric and set-theoretic definitions of a model, which consists a cone, two spheres and blends between the cone and the two spheres (Fig.3.25). This description is written in an input language called BLINP, which was written in FORTRAN by Mr Andrew Wallis. The operator symbols are defined by the language, while the names of shape elements are chosen by the user. One of the strengths of this language is that it allows the user to use free mathematical form in defining implicit polynomials; in other words, it can be used to define any half-space which can be specified in the form of implicit polynomial inequalities, no matter how complicated it is.

The main purpose of this input language was for experimenting with various surfaces. In the development of the solid modelling system, several complicated models have been constructed in this language.

As it is simple and flexible, no transformation instructions are provided and the same name is not allowed to be used on both sides of an assignment. However it allows the user define colour and viewing parameters for generating shaded pictures.

Example 2

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
C
C       Example of Some Primitives.
C
C ----- A cube, scaled and coloured.
C
        CUBE = SCL(*CUBE,50)
        CUBE = STCB(CUBE,3)
C
C ----- A sphere above the cube.
C
        SPHERE = *SPH1(0.,100.,0.,50)
        SPHERE = STCB(SPHERE,5)
C
C ----- A cylinder beside the cube.
C
        CYLINDER = *CYLX(50)
        BOUND = MOVX(*IHSX,200).INT.MOVX(OHSX,100)
        CYLINDER = CYLINDER.INT.BOUND
C
C ----- Union of the three defined primitives.
C
        OBJECT = CUBE.UNI.SPHERE.UNI.CYLINDER
C
C ----- Rotate the object first round Y axis, then X axis.
C
        OBJECT = ROTX(ROTY(OBJECT,-45),-30)
C
C ----- Loop, generate two extra copies.
C
        OBJECT = MOVX(MOVY(OBJECT,-45),-200)
C
        OBJECT1 = OBJECT
C
        LOOP = 2
        OBJECT = MOVY(MOGX(OBJECT,120),150)
        OBEJCT = ROTY(OBJECT,15)
        OBJECT = ROTZ(OBJECT,30)
        OBJECT1 = OBJECT.UNI.OBJECT1
        ENDLOOP
C
C ----- Naming the output object.
C
        OUTPUT = OBJECT1
C
        END
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

This example is a description of three groups of primitives, each of them consists of a cube, a cylinder and a sphere (Fig.7.4). This input language is written by the author for the purpose of generating models quickly so that more effort could be paid to experiments on other parts of the solid modelling system. It uses syntax rules similar to

the FORTRAN language. For example, the name of a shape element can be used any where for as many times as needed, as long as its first appearance is well defined. This allows a complicated model to be described relatively concisely. It also provides some commonly used primitives such as those shown in the example, though it allows the user to built up his own primitives.

One advantage of this language is that it provides transformation facilities and loops for modifying, manipulating and duplicating shape elements already defined, as illustrated by the example. Therefore, it can generally handle very complicated assemblies.

But, as free format of mathematical formulation is not supported in this input language, it is not suitable for describing high order surfaces.

Example 3

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Global declarations. Sets to put
; the block and hole in. Points are also needed to
; define the corners of the blocks.

Sets {block,hole}
Points {lowcorn, highcorn}

; Define a function that creates a cuboid from the two
; Points on the ends of its leading space diagonal.

FUNCTION box (minpt: Point, maxpt: Point): Set

; Local variables. Six Sets for the faces, and one for
; the resulting cuboid.

        Sets {minxf,minyf,minzf,
                maxxf,maxyf,maxzf}
        Sets {cuboid}

; We also need the coordinate directions
; (and their negatives).

        Points {x_vec,y_vec,z_vec}

; Start of the executable code. Use the built in space
```

```
; function to define the faces from Points in them and
; the vectors defining the coordinate directions.

        {x_vec:=pt(1,0,0)
         y_vec:=pt(0,1,0)
         z_vec:=pt(0,0,1)

; Define the maximum three faces

        maxxf:=space(x_vec,maxpt)
        maxyf:=space(y_vec,maxpt)
        maxzf:=space(z_vec,maxpt)

; Negate the vector directions

        x_vec:=-x_vec
        y_vec:=-y_vec
        z_vec:=-z_vec

; Define the minimum faces

        minxf:=space(x_vec,minpt)
        minyf:=space(y_vec,minpt)
        minzf:=space(z_vec,minpt)

; Now define and return the cuboid.

        cuboid:=minxf & minyf & minzf & maxxf &
                maxyf & maxzf
        RETURN (cuboid)
        }

; MAIN PROGRAM

; Use the function to build the block.

        lowcorn:=pt(1,1,1)
        highcorn:=pt(2,2,2)
        block:=box(lowcorn,highcorn)

; Set the block to be colour 1

        block:=colour(block,1)

; Make the hole, and give it colour 2.

        lowcorn:=pt(1.25,1.25,1.25)
        highcorn:=pt(1.75,1.75,1.75)
        hole:=box(lowcorn,highcorn)
        hole:=colour(hole,2)

; Subtract the hole from the block and record the result.

        block:=block~hole
        WRITE('my_output_file',block)
```

```
}
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

The model described by this example is a block with a hole in. This example is written in an input language called SID, originated by Dr Adrian Bowyer. It was written mainly for describing models to be evaluated by DORA (A CSG solid modeller written by Dr John R Woodwark.), which handles solid models constructed purely from planar half-spaces. SID can also be used to describe models consisting curved surfaces defined by implicit polynomials.

It is rather like the PASCAL language in both syntax rules and structures. It has a rich set of operators, which allow for many sophisticated operations.

Though it does not provide ready built primitives, it gives the user convenient means to built up his own primitives, which can be used easily in subsequent definitions. Some commonly used primitives are constructed in separate files. Using the INCLUDE command provide in this language, the user can combine these primitives into his file. This reduces the effort needed in constructing a complicated model. It also provides transformation facilities. Generally speaking, SID is a much more sophisticated input language than the other two, and is particularly suitable for CSG solid modelers handling models defined by half-spaces.

4.4 CONVERSION FROM USER DESCRIPTION TO FORMATTED INPUT

The function of an input language is not only to translate the text file statement by statement into another form, like that between natural languages, but also to perform certain evaluations. These operations are needed to generate a compact, uniform and well structured output.

Some basic operations that are carried out by the input language are briefly explained here.

4.4.1 Planar Half-Space Verification

Planar half-spaces are the basic and simplest shape elements used in the construction of a model. They are also the basic elements used to specify any non-planar half-spaces. Therefore, it is important to keep data on these correct, simple, compact and quick to use.

A planar half-space is defined by the user. At least four values are needed to specify a plane. Here A_1, A_2, A_3 and A_4 are used for them. If they are normalised, that it to say, if they meet the requirement

$$A_1^2 + A_2^2 + A_3^2 = 1 \tag{4.1}$$

then the vector $N = [A_1\, A_2\, A_3]$ represents the normal direction of the plane and A_4 represents the distance between the plane and the origin.

There are several ways to determine these four values. Some of them are described as follows.

(a). The user can give the precise values directly. This is easy for planes perpendicular to one of the axes, or in a simple relative position to the co-ordinate system, such as positions at angles of 45 or 60 degrees. But for planes in other orientations, this becomes

difficult.

Suppose that the values of a vector and the distance are given as A_1', A_2', A_3' and A_4', the normalisation is performed as follows

$$A_1 = \frac{A_1'}{\sqrt{A_1'+A_2'+A_3'}} \tag{4.2a}$$

$$A_2 = \frac{A_2'}{\sqrt{A_2'+A_2'+A_3'}} \tag{4.2b}$$

$$A_3 = \frac{A_3'}{\sqrt{A_3'+A_2'+A_3'}} \tag{4.2c}$$

$$A_4 = A_4' \tag{4.2d}$$

where, of course, A_1', A_2' and A_3' must not be all zero.

(b). Instead of specifying them precisely, the user can relatively easily fix the position and orientation by giving the angles between the plane normal vector and the axes, and the distance between the plane and the origin. Suppose that the angles are given in degrees as D_a, D_b and D_c (in fact, two angles are enough to determine the third angle, but the third value is necessary to determine the direction of the vector) and the distance in length D_d, then the calculations for the actual values are straight forward:

$$A_1 = \cos D_a \tag{4.3a}$$

$$A_2 = \cos D_b \tag{4.3b}$$

$$A_3 = \cos D_c \tag{4.3c}$$

$$A_4 = D_d \tag{4.3d}$$

If these angles were given accurately, there is no need to perform normalisation, because the cosine function ensure normality. However, if these angles are not accurate or they are inconsistent, it is necessary to perform the normalisation to make sure. This is a more convenient way than the first one but still, if the plane is in an arbitrary orientation, it would be difficult for the user to fix these angles easily.

(c). Another way to specify a plane is to give the co-ordinates of three non-co-linear points in space. Suppose these three points are given as (X_1, Y_1, Z_1), (X_2, Y_2, Z_2) and (X_3, Y_3, Z_3). The four values can be calculated using the following formula

$$A_1 = \begin{vmatrix} Y_1 & Z_1 & 1 \\ Y_2 & Z_2 & 1 \\ Y_3 & Z_3 & 1 \end{vmatrix} \tag{4.4a}$$

$$A_2 = \begin{vmatrix} Z_1 & X_1 & 1 \\ Z_2 & X_2 & 1 \\ Z_3 & X_3 & 1 \end{vmatrix} \tag{4.4b}$$

$$A_3 = \begin{vmatrix} X_1 & Y_1 & 1 \\ X_2 & Y_2 & 1 \\ X_3 & Y_3 & 1 \end{vmatrix} \tag{4.4c}$$

$$A_4 = \begin{vmatrix} X_1 & Y_1 & Z_1 \\ X_2 & Y_2 & Z_2 \\ X_3 & Y_3 & Z_3 \end{vmatrix} \tag{4.4d}$$

and then normalised. The direction of the plane is determined implicitly by the sequence of the three points appearing in the calculation [16]. (A convention is used such as the points appearing in clockwise order when viewed from solid to air.)

As we can see from the above discussion, none of these methods is absolutely better than the others. When defining a new model, the first and second method probably are more convenient. If the model is to be constructed based on some data obtained by measuring a real object, the last method is probable the best one. Other methods (such as giving the angles of the plane normal and the position of a point that lies on the plane, or defining a plane in a fixed position and orientation and moving it to required place by means of transformation) can also be used if more appropriate. Whatever method is used, the above operations must be carried out to make sure that each of the planes is absolutely correct

before their further use.

4.4.2 Transformation

Transformation operations are required in both model constructions and analysis; for example, when constructing a complicated model from primitives, a model of an assembly or when modelling robotic movement. Also, when viewing a model or cutting the surfaces of the model, transformation is needed to place the model in a proper position and orientation in relation to the display device or the co-ordinate systems on the NC machine tool.

While other transformations can be performed by a solid modeller during model evaluation, those transformations needed to construct the model should not be left until then, otherwise the data would be less compact and would complicate the subsequent evaluation. It is more appropriate if these transformations are performed by the input language processor.

One very important characteristic of the solid modelling system described in this thesis is that all complicated geometric entities are constructed from planar half-spaces. The relative position between planar half-spaces will completely determine the size and shape of any subsequently defined surfaces. Therefore the definitions of all kinds of half-spaces other than the original planes are axis-independent; that it to say, in constructing a higher order surface using the multi-stage definition method, the user does not have to worry about their position relative to the axes, he only needs to consider the relations between the defined surface and the defining half-spaces, which, of course, can be curved half-spaces as well as planes. This feature simplifies the transformation operation a great deal. Some of the basic transformation operations are described in what follows.

4.4.2.1 Translation

Translations is the simplest transformation operation. It changes the position of a shape element in space without altering its orientation. In this solid modelling system,

translation is done at the planar half-space level. As we mentioned above, not only will this translate the positions of these planes, but it also effectively translates all the half-spaces defined by these planes. Since the relative position between planar half-spaces remain unchanged after translation, the shape of the shape element, which can be a high order half-space, will not be changed.

In translating a plane, the instruction can be given in two ways. One is to give the relative distance to move and the direction. The other is to specify the destination position and let the computer to work out the amount of movement and direction. (a point on the plane has to be given to be used as a reference point in the second case.)

As we discussed earlier, a plane is defined by four values A_1, A_2, A_3 and A_4. Translating a plane in space is to move it in any direction without rotation. This is in fact to change the value of A_4, which is the distance between the plane and the origin. Supposing that T is the relative distance to move. If the plane is to be moved along its normal direction, the calculation is straightforward

$$A_4' = A_4 + T \tag{4.5}$$

where T can be negative if moving in the opposite direction. For translating a plane in an arbitrary direction represented by the unit vector $\mathbf{N}_T$, the following general equation can be used (Fig.4.1).

$$A_4' = A_4 + (\mathbf{N}_T \cdot \mathbf{N})T \tag{4.6}$$

where $\mathbf{N}$ is the surface normal of the plane. From this equation we can see that when the translating direction is the same as the surface normal, it is identical to equation (4.5); when these two vectors are perpendicular to each other, the plane is unchanged. When translation along one axis is required, the following simplified equation can be used instead.

$$\textit{Along X axis:}\quad A_4' = A_4 + A_1T \tag{4.7a}$$

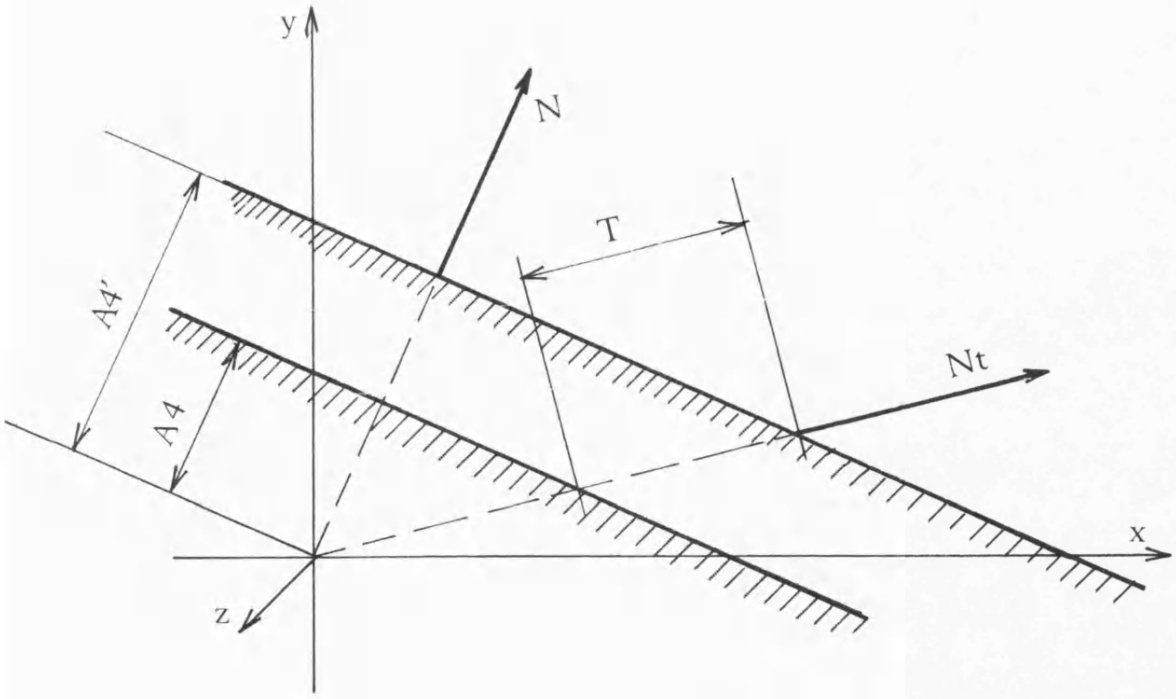


Fig.4.1 Translation of a Plane

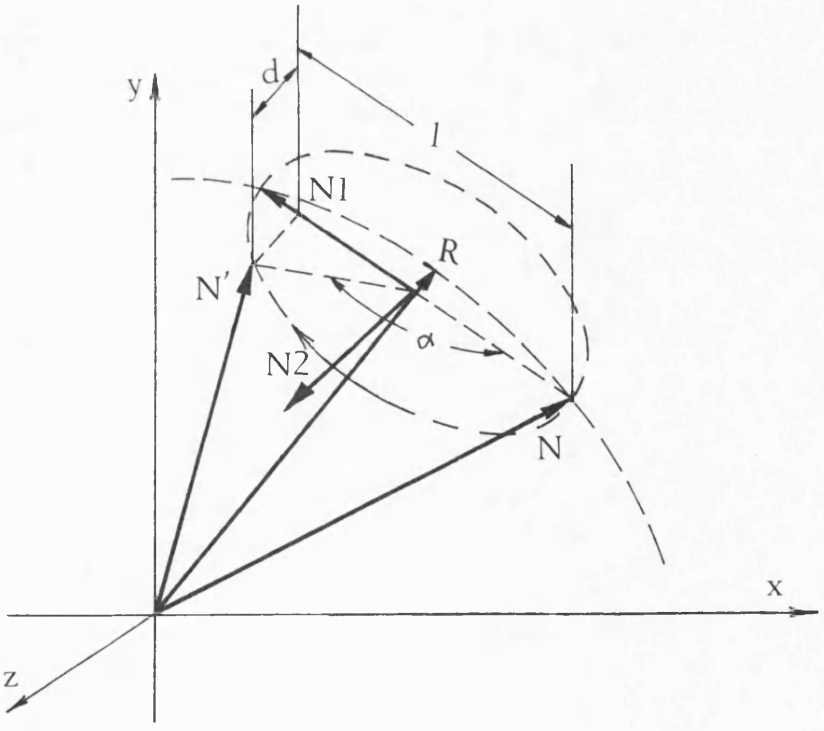


Fig.4.2 Rotation of a Plane

Along Y axis: $A_4' = A_4 + A_2 T$ (4.7b)

Along X axis: $A_4' = A_4 + A_3 T$ (4.7c)

In practice, the user does not have to concentrate on the translation of every single plane, he only needs to specify the translation distance and direction for a high order half-space, a part of the model, or the entire model. The computer searches for all the related planar half-spaces and carries out the required translation operation at the plane level. To the user, it seems that he is operating on the model or shape element directly.

4.4.2.2 Rotation

Just like translation, rotations are also performed at the planar half-space level. The shape elements defined by these planes will automatically be rotated the same amount. Rotation operations also have the property that the relative positions between planes will remain the same as before the operation, hence keeping the shape of rotated element unchanged.

The simplest rotation is about a single axis. In such a rotation, the distance A_4 between the plane and the origin will not be changed. Also, since the co-ordinates of the shape element will not be changed along the axis about which the rotation is carried out, we need only consider the other two axes. If α is the angle through which a plane is to be rotated, the following formula will give the new value of relevant values of A_1, A_2 and A_3 after rotation about the OZ, OX and OY axis respectively

About OZ axis: $$\begin{bmatrix} A_1' \\ A_2' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$ (4.8a)

About OY axis: $$\begin{bmatrix} A_2' \\ A_3' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} A_2 \\ A_3 \end{bmatrix}$$ (4.8b)

About OX axis: $$\begin{bmatrix} A_3' \\ A_1' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} A_3 \\ A_1 \end{bmatrix}$$ (4.8c)

Rotations about a line passing through the origin but in an arbitrary orientation are more difficult. The three axes must be all considered and more complicated calculations are involved.

For a given plane with unit normal vector $\mathbf{N} = [A\ B\ C]$ and a line with the unit direction vector $\mathbf{R}$, the new values of the unit normal $\mathbf{N}' = [A'\ B'\ C']$ of the plane after the rotation about the line $\mathbf{R}$ through an angle α, as shown in Fig 4.2, can be calculated as follows

$$\mathbf{N}' = \left[A'\ B'\ C'\right] = \left[1\ l\ d\right]\begin{bmatrix}\mathbf{N}\\ \mathbf{N}_1\\ \mathbf{N}_2\end{bmatrix} = \left[1\ l\ d\right]\begin{bmatrix}A & B & C\\ A_1 & B_1 & C_1\\ A_2 & B_2 & C_2\end{bmatrix} \tag{4.9a}$$

where

$$l = (1-\cos\alpha)\sqrt{(1-(\mathbf{R}\cdot\mathbf{N})^2} \tag{4.9b}$$

$$d = \sin\alpha\sqrt{(1-(\mathbf{R}\cdot\mathbf{N})^2} \tag{4.9c}$$

$$\mathbf{N}_1 = \left[A_1\ B_1\ C_1\right] = (\mathbf{R}\cdot\mathbf{N})\mathbf{R} - \mathbf{N} \tag{4.9d}$$

$$\mathbf{N}_2 = \left[A_2\ B_2\ C_2\right] = \mathbf{N}\times\mathbf{R} \tag{4.9e}$$

The line $\mathbf{R}$ can be in any orientation and the angle α can either be positive or negative in relation to $\mathbf{R}$. The new values $\mathbf{N}'$ is also a unit vector as long as $\mathbf{R}$ and $\mathbf{N}$ are unit vectors.

Rotations about an arbitrary line not passing through the origin involves changing both the direction of the plane and the distance between it and the origin. Such operations can be performed by combining translations and rotations described above. For example, such an operation can be done by firstly translating the plane and the line to a position such that the line passes through the origin, secondly, performing the rotation through a given angle at the new position and finally, translating the plane by the value that is the negative

of that of the first translation. The resultant plane will be in the required position and orientation. If each of the individual transformations is represented by a homogeneous matrix $T_1, T_2, T_i, ..., T_n$, by multiplying them together as

$$T = T_1 T_2 T_i \cdots T_n \tag{4.10}$$

the entire transformation can be represented by the result matrix T [10] [80][81].

4.4.2.3 Scaling

In the system described in this thesis, all geometric entities are constructed from planar half-space and constants. As with translation and rotation, the scaling operation is also required to be performed at the planar half-space level. Translation and rotation is relatively simple, because the relationships of distance and orientation between planes are not changed afterwards. However, for scaling operations, since the size, shape and position of shape elements are to be changed, we have to modify the relative positions and orientations between their defining planar half-spaces.

The simplest scaling is purely to change the size of shape elements and is hence called *overall scaling*. For such operations, only the surface normal is modified according to the scaling facto F (Fig.4.3). The new surface normal of a plane is

$$\mathbf{N}' = \frac{\mathbf{N}}{F} = [\frac{A_1}{F} \ \frac{A_2}{F} \ \frac{A_3}{F}] \tag{4.11}$$

The distance A_4 is unchanged, so that it will give the same potential value at the origin, which is the reference point for such scaling.

Scaling may also be required to be carried out in only one direction, with a reference point other than the origin. This is much more complicated than other transformations described so far, because it changes all the four coefficients of a plane. After such an operation, in general, a plane is also translated and rotated, and its potential values are no longer guaranteed to represent true distance, because its normalisation is destroyed. The following

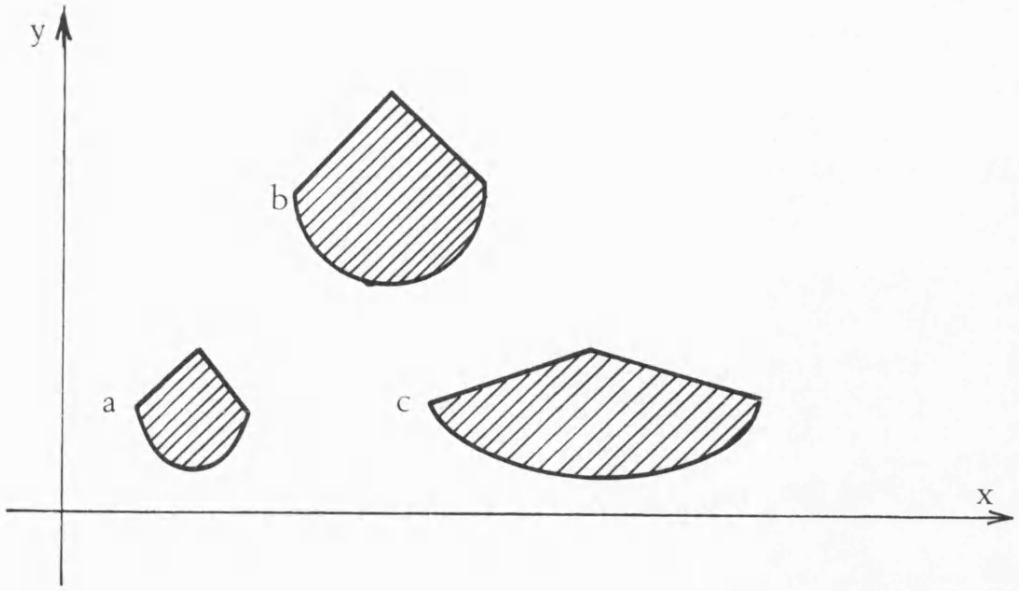


Fig.4.3 Overall Scaling (b) and Uni-Directional Scaling (c)

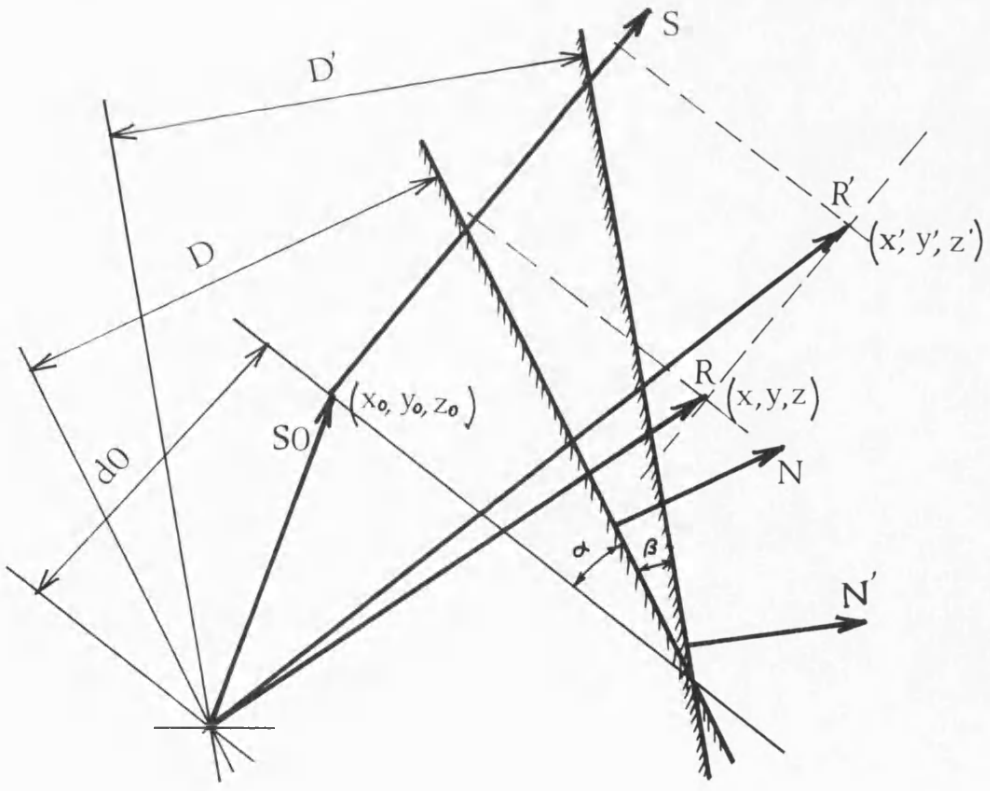


Fig.4.4 Uni-Directional Scaling of a Plane

describes how the coefficients of scaled planes can be calculated.

Suppose that $\mathbf{N}$ is the surface normal of a plane and $\mathbf{R}$ is an arbitrary vector in space, and the plane is scaled in direction $\mathbf{S}$, which is a unit vector, from a reference point (X_0, Y_0, Z_0) by a scaling factor F. If $\mathbf{R}$ becomes $\mathbf{R}'$ and $\mathbf{N}$ becomes $\mathbf{N}'$ after scaling (Fig.4.4), then we have

$$\mathbf{N}\cdot\mathbf{R} = \mathbf{N}'\cdot\mathbf{R}' \tag{4.12a}$$

or

$$A_1X + A_2Y + A_3Z + A_4 = A_1'X' + A_2'Y' + A_3'Z' + A_4' \tag{4.12b}$$

so that the potential value at (X', Y', Z') will be the same as at (X, Y, Z), in other words, if a point is on the plane before the scaling, it will also on the plane after the scaling, though its position in space is changed. From Fig.4.4, the relationship between $\mathbf{R}$ and $\mathbf{R}'$ can be expressed as

$$\mathbf{R} = \mathbf{R}' - f\,[(\mathbf{R}' - \mathbf{S}_0)\cdot\mathbf{S}]\mathbf{S} \tag{4.13}$$

where $\mathbf{S}_0$ is the vector of the reference point and $f = 1 - \frac{1}{F}$. From this equation, we can calculate the relationship between each of the coordinates as follows.

$$X = X' - S_x f\,[(S_x X' + S_y Y' + S_z Z') - d_0] \tag{4.14a}$$

$$Y = Y' - S_y f\,[(S_x X' + S_y Y' + S_z Z') - d_0] \tag{4.14b}$$

$$Z = Z' - S_z f\,[(S_x X' + S_y Y' + S_z Z') - d_0] \tag{4.14c}$$

where $[S_x\ S_y\ S_z] = \mathbf{S}$ and $d_0 = S_x X_0 + S_y Y_0 + S_z Z_0$, which is the distance from the reference point to the origin in direction $\mathbf{S}$. From equations (4.12) and (4.14), the coefficients of the plane after scaling can be calculated as

$$A_1' = A_1 - f S_x (A_1 S_x + A_2 S_y + A_3 S_z) \tag{4.15a}$$

$$A_2' = A_2 - fS_y(A_1S_x + A_2S_y + A_3S_z) \tag{4.15b}$$

$$A_3' = A_3 - fS_z(A_1S_x + A_2S_y + A_3S_z) \tag{4.15c}$$

$$A_4' = A_4 + fd_0(A_1S_x + A_2S_y + A_3S_z) \tag{4.15c}$$

If the angle between the original surface normal **N** and the scaling vector **S** is α, the angle β between **N** and **N'** can be calculated from the following equation.

$$\tan\beta = \frac{(F-1)\sin\alpha\cos\alpha}{\cos^2\alpha + F\sin^2\alpha} \tag{4.16}$$

From this equation we can see that the plane is not rotated if F is 1 or α is a right angle. If D is the *true* distance between the plane and the origin before scaling, and D' is the *true* distance after scaling, the ratio $f_n = \frac{D}{D'}$ can be calculated using the following equation

$$f_n = \frac{D}{FD\cos\beta + (1-F)[D\sin\alpha\sin(\alpha+\beta) + d_0\cos(\alpha+\beta)]} \tag{4.17}$$

When d_0 is 0, in other words, if the reference point is at the origin, or the distances are calculated from the reference point, f_n represents the ratio between the new surface normal **N'** and the old one, **N**. We can see from the above equation that $f_n = 1$ only when $F = 1$ or $\alpha = 90$ degrees, and that the normality of the plane is in most cases changed.

As we discussed earlier, because the potential values of normalised planes represent true distances, they are convenient to use. For example, the user has a better understanding of the size and dimensions of a model when describing it, and also, it is much simpler to calculate the the illumination shade for a normalised plane. But the normality has to be changed in order to carry out the scaling operation. To satisfy both of these requirements, we can multiply each plane with one more factor, A_n, which can be called the *normalising factor* and can be calculated using the following equation. (This technique was used in the input language processor SID by Dr Bowyer.)

$$A_n = \sqrt{A_1'^2 + A_2'^2 + A_3'^2} \tag{4.18}$$

Then, the scaled plane can be normalised as follows.

$$A_1'' = \frac{A_1'}{A_n} \tag{4.19a}$$

$$A_2'' = \frac{A_2'}{A_n} \tag{4.19b}$$

$$A_3'' = \frac{A_3'}{A_n} \tag{4.19c}$$

$$A_4'' = \frac{A_4'}{A_n} \tag{4.19d}$$

hence we have a normalised plane equation

$$H_{1a} = A_1''X + A_2''Y + A_3''Z + A_3'' \tag{4.20}$$

In polynomial definition, the following form of the plane should be used

$$H_{1n} = A_n(A_1''X + A_2''Y + A_3''Z + A_3'') \tag{4.21}$$

Other kinds of transformation operations are also useful [6], such as reflection, bending, twisting, distortion, plastic deformation, perspective, projection and so on. Their integration with a solid modelling system would greatly increase its power, flexibility and range of applications.

[109]

4.4.3 Set-Theoretic Operator Rationalisation

There are three set-theoretic operations used to describe the construction of the model from half-spaces. They are union ($A \bigcup B$), intersection ($A \bigcap B$) and difference ($A - B$). Union and intersection are two basic operators, while difference can be replaced by other more basic operations: the intersection and complement operation

$$A - B = A \bigcap B' \tag{4.22}$$

where B' is the complement of B.

Though difference is theoretically redundant, it is much more convenient and intuitive to use in practice than the other form of operation, particularly during the construction of the model.

However, after the definition stage, the set-theoretic operations are dealt with entirely by the solid modeller. To make the data structures and operations simpler, it is better to use the other form of the difference operator, so that only two different operators are needed. The complement of the operand can be done by reversing the direction of the half-space. In other words, changing the solid side into air and changing the air side into solid. This can be done simply by exchanging all the plus and minus operators used in defining that half-space. Another way to do this is to multiply each half-space in the set theoretic operation expression by -1. This is easy to do and absolutely reliable, but needs more storage to accommodate the extra operators and operands.

4.4.4 The Converted Form

The user input for a model may be written in different input languages, but the output of these languages should take the same form if they are to be used by the same modeller. In addition to the geometric information, the output should also have the same data structure and format.

To meet this requirement, data re-organisation (some of which have just been discussed) must be performed during the translation. In the end, the data should occupy minimum space, carry all the information and have a format compatible with the modeller.

There are four basic parts in the converted form: the planar half-spaces, the constants, the polynomial definitions and the set theoretic expressions. Other information such

as colour and viewing parameters are also included. Appendix 1 gives an example of the data and format of a converted, or a formatted, input file.

4.5 MODIFICATION OF THE MODEL

As was discussed earlier, a model is completely described by a text file for the input language in use. During the model's construction no use was made of any interactive graphic aid. However, before submitting the model definition to the modeller for any serious evaluation, it is important to know two things:

(1). The user needs to be sure that the model represents the desired shape. The shape of a model (very often a new product) is usually defined initially by some sketches on paper, two-dimensional engineering drawings, or even straight from the user's imagination. If the model becomes rather complicated, it is difficult for the user to know what it looks like just from the language text. Furthermore, subsequent evaluations and operations, such as generating shaded pictures or NC cutting codes, normally take a considerable time. It is worth knowing the shape of the model before these processes are carried out to avoid mistakes.

(2). The user needs to know that the shape, range and extent of the blend surfaces are right. The controlling constants [Chapter 3] of blend surfaces, unlike other constants such as the radius of a cylinder or the angle of a cone, have geometric meanings which are hard to grasp. They vary from one blend to another. It is very difficult to decide their value correctly without any feedback, and even so, since their effect is non-linear, it usually needs several modifications to get it right.

These two practical requirements cannot be satisfied without using graphics. Generally speaking, a solid model can be constructed interactively or using a text file. The former gives the user quick feedback and he knows what he is doing, but as the feedback also needs computing time (even for simple models), interactive construction of a model may be slower than using a text file. The latter may be quick for constructing models having relatively complicated but low order geometry, but it has little advantage for blend

surface definition and other complicated definitions that need feedback.

If these two techniques for the construction of a model are combined, it will certainly be possible to do it better and quicker.

Section views of half-spaces, of parts or whole models can be produced fairly quickly. With such graphic images, the user can conveniently view his design. In the system described in this thesis, programs are designed to take complete or partially complete model description files and to generate line drawings for single half-spaces or sub-models. This facility enables the user to construct a correct model before subsequent operations are carried out using it. When viewing the model, the user does not have to look at the whole geometric definition, he can view whatever he is interested in.

The interactive cross-section program takes the same input file as the solid modeller does. It operates on the same information but does not generate the same output. It is used to check, transform and modify the values of constants and planar half-spaces, the position and orientation of the model, or part of it.

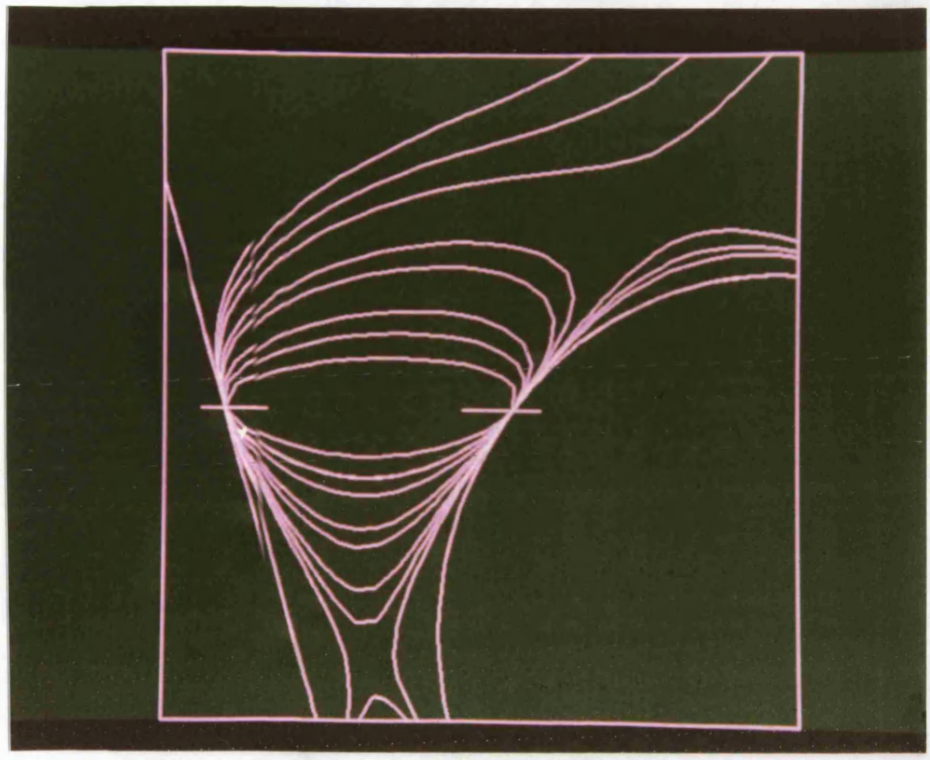
The interactive program allows the following operations.

1. Checking

the definition of a model can be checked at its various stages of definition. This is important as it allows the user to make sure that a half-space is right when it is used to define another more complicated one.

2. Determination of constant values

The value of a control constant for a blend surface is difficult to fix without interactive feedback. This program enables the user to look at the effect of the constant quickly, and hence modify it more conveniently (Fig.4.5).



Fiig.4.5 Modification of a Blend

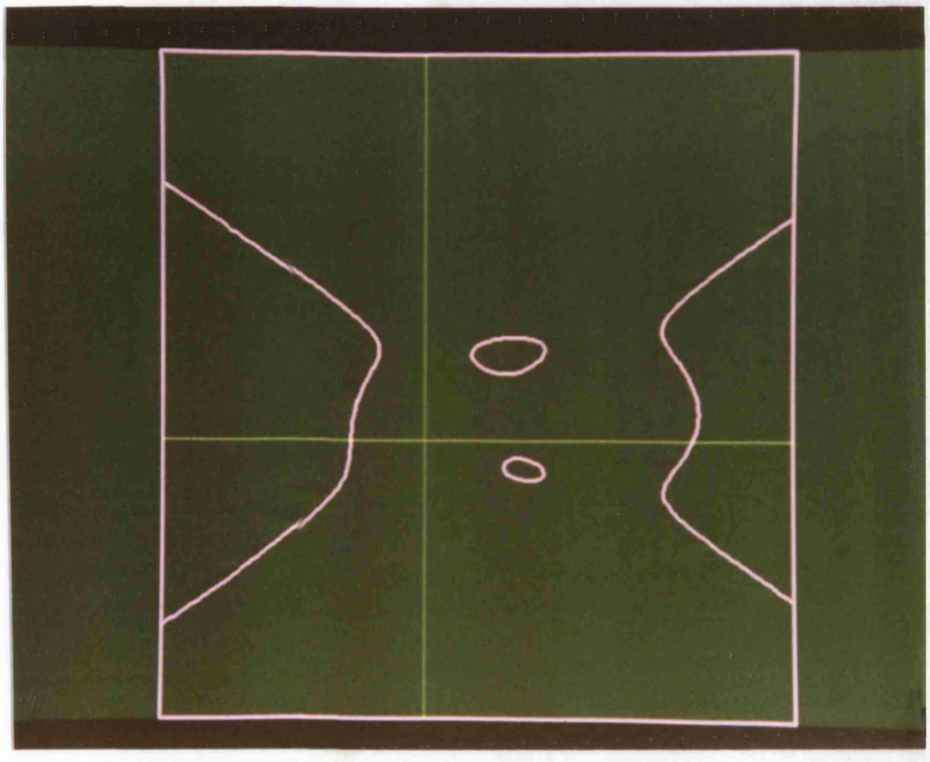


Fig.4.6 Viewing of a Single Surface

3. Modification of Planar Half-Spaces

By modifying the position ad orientation of planes, it is possible to change the shape and size of curved surfaces, particularly, when planes are used as biasing surfaces (Fig.3.21), because planes are the basic variables in the definition of high order half-spaces.

4. Viewing of individual surfaces

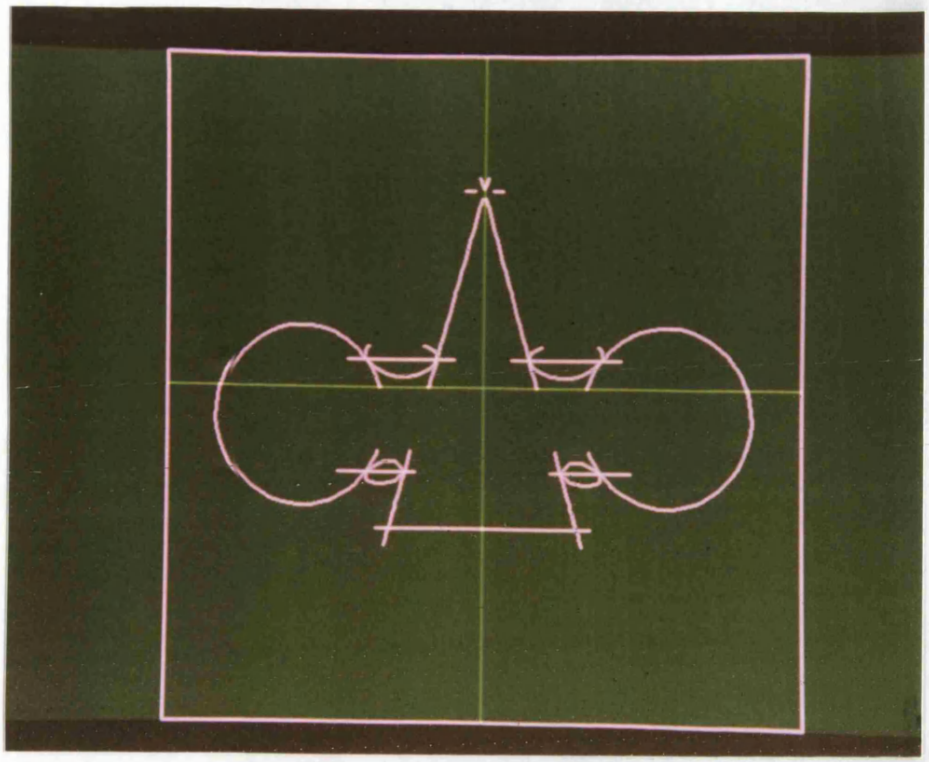
In the shaded, three dimensional output of the solid modeller, it is difficult to look at individual surfaces, especially those which are not used in the output. For example, a ranging surface, which is geometrically defined, will not appear on the shaded picture of the final model. In the interactive program, any surface that is defined in the model definition file can be selected and checked on the screen. Fig.4.6 shows a cross-section curve of the whole blend surface between a cone and a sphere (Fig.3.25) within the object space.

5. Transformation

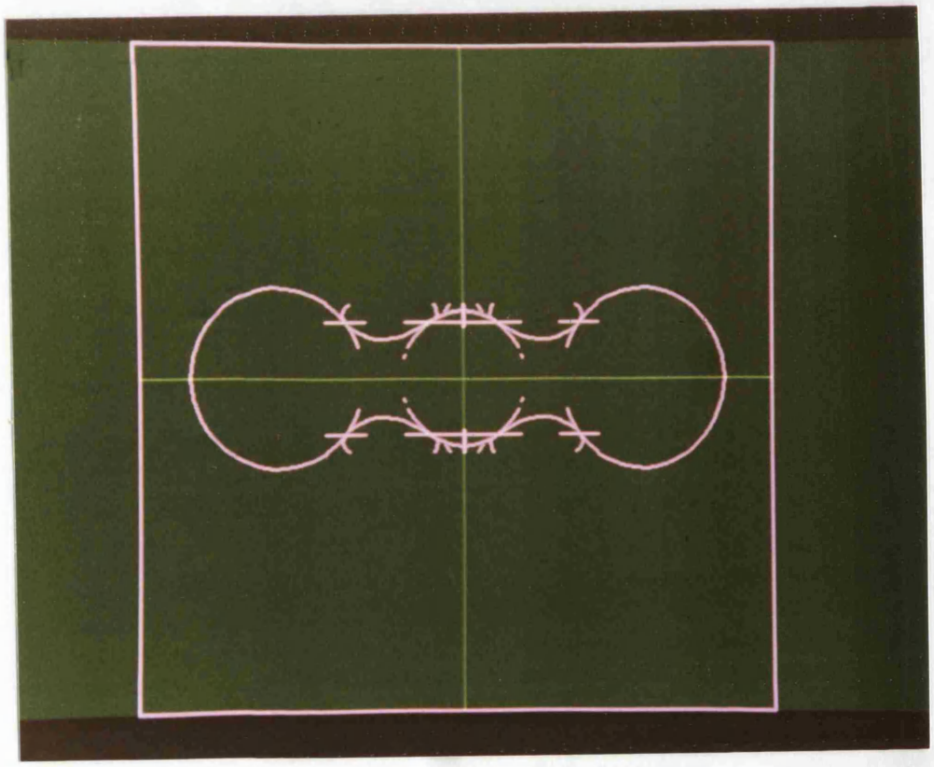
The shape of the model can be partially or entirely moved, rotated and scaled. This can be done more intuitively with the aid of graphical feedback, though these operations are available in the construction of the text description of the model.

6. Section view

This is in fact the main visual aid provided in the interactive program. By defining the section position and transforming the model, or part of it, into a proper position, the model can be viewed (Fig.4.7) more quickly than a shaded picture view of the model can be computed.



(a)



(b)

Fig.4.7 Section Views of a Model

The definition of a model is originally written in a text file. Because of its structure, it is difficult to convert a feedback-modified description back to its original text form. But the result of modification can be re-written to the original file which was used as the input for modification, so that this file remains unchanged in form, but contains different values for the modified model.

If the original text file is to be renewed for documentary purpose, the modification can be recorded for the user's reference.

CHAPTER 5

SUBDIVISION AND PRUNING TO IMPROVE EFFICIENCY

As was discussed in Chapter 2, two of the main problems with the CSG scheme are the lack of locality of the surfaces and redundant operations on the operation tree which defines the model as a single set-theoretic expression.

These two complexity problems can make calculations slow and some of the subsequent operations quite difficult and inefficient. For example, a model can be constructed using hundreds of half-spaces with a set-theoretic expression thousands of terms long. During evaluation (using ray tracing to produce a picture, for example) intersections of the rays and the object's surface have to be calculated. Since any of the half-spaces may contribute to the object's surface, it is time consuming to decide which half-space should be considered in the intersection calculation, as interrogating all the half-spaces and going through the entire length of the definition tree involves a considerable amount of work.

When faced with such a problem, it is worthwhile considering if it can be attacked with a *Divide and Conquer* algorithm. By breaking down the entire burden spatially, half-spaces could be associated with the relevant areas on the object's surface, making operations much more direct and quick [106] [109]. In the system described in this thesis, two integrated techniques, *sub-division* and *pruning*, were used. They have proven very effective, and it was these techniques that made the CSG scheme both theoretically more robust and practically more useful.

5.1 SUBDIVISION OF THE OBJECT SPACE

Originally, the defined model is surrounded in a fixed cuboidal space, the size of which is decided by the user in the model's definition text file, or in a command file that contains instructions to run the program. This initial space is called the *object space* and it usually takes the form of a cuboid with its six faces parallel to X-Y, Y-Z and Z-X planes respectively.

The evaluation of the model is done within the object space; in other words, the object space is evaluated against every element on the CSG tree and is hence spatially divided into a tree of solid and air regions. The resulting solid part of the object space represents most of the body of the model. The rest of the model is those parts of it which lie close to its surface. These parts are all leaves of the division tree, and are divided regions that are part solid, part air (Fig.5.1).

To obtain this result, the object space has to be compared with every single half-space on the CSG tree to see if the half-space lies inside, or outside, or crosses the object space, and if it crosses, how it contributes to the final model surface.

The subdivision method is probably the most reasonable approach to this problem. By recursively dividing the object space into many smaller spaces, which are called *sub-spaces*, the whole burden of the heavy evaluation of the original object space is now reduced to lots of sub-problems at each level of division, which can be resolved separately, and much more easily.

The sub-division method serves to achieve two beneficial effects in the solution of the complexity problems. Firstly, by reducing the size of the object spaces and subsequent sub-spaces, fewer half-spaces will lie inside or partially inside each of them. This reduces the effort in checking and calculation between half-spaces in each individual sub-space. For example, there are seven half-spaces crossing the object space abcd (Fig.5.2), but only three of them cross sub-space a'b'c'd'. Secondly, when the

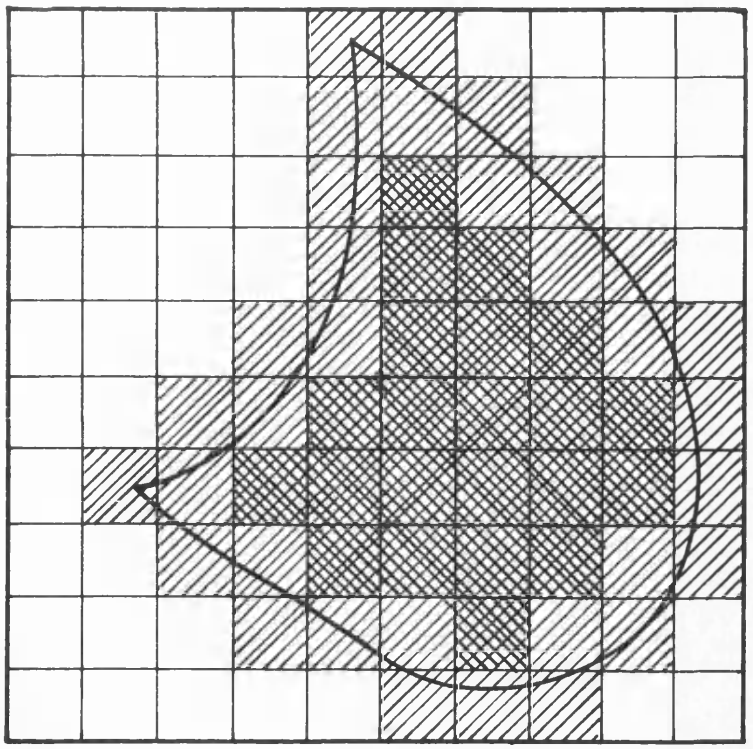


Fig.5.1 Sub-division of the Object Space

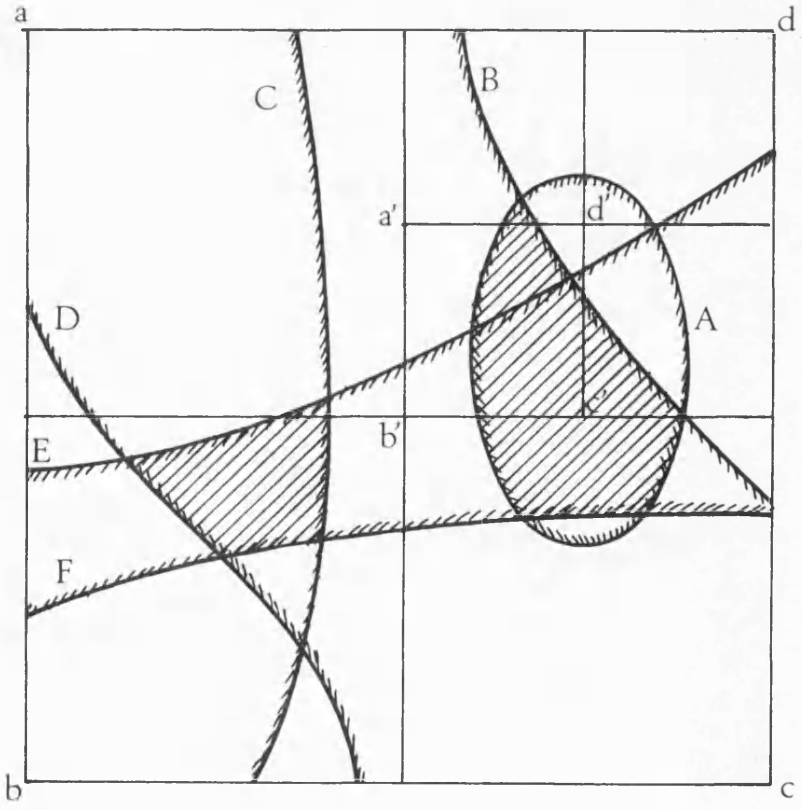


Fig.5.2 Sub-Spaces

space becomes smaller, the relationship between a half-space and a sub-space becomes simpler, such as half-space A in Fig.5.2. This is because the calculation between a curved half-space and an sub-space is more difficult than that between the same half-space and an sub-space of smaller size. Since the order of a half-space can not be changed during the evaluation, dividing the object space into smaller areas is therefore the natural way of handling this problem.

Using the recursive division technique, the object space is first divided in to a certain number of sub-spaces. They are checked before further division. If a subspace contains no surface, no further division and evaluation is needed. Otherwise, the subspace is further divided in the same manner into smaller ones, which are checked and divided again until they become empty or reach the smallest permitted subspace size defined by the user. It is necessary to specify a limit to prevent the system attempting to divide some subspaces for ever. For example, if a corner formed by several surfaces exists in a subspace, no matter how many smaller sub-space it is further divided into, one of them will still contain this corner. If the decision on a subspace's further division is judged only by the number of half-spaces it contains, it would be difficult to handle this corner problem.

After division, all the partially empty final sub-spaces are evaluated separately. The divided model may be evaluated in *better than linear* time against the model's complexity [106].

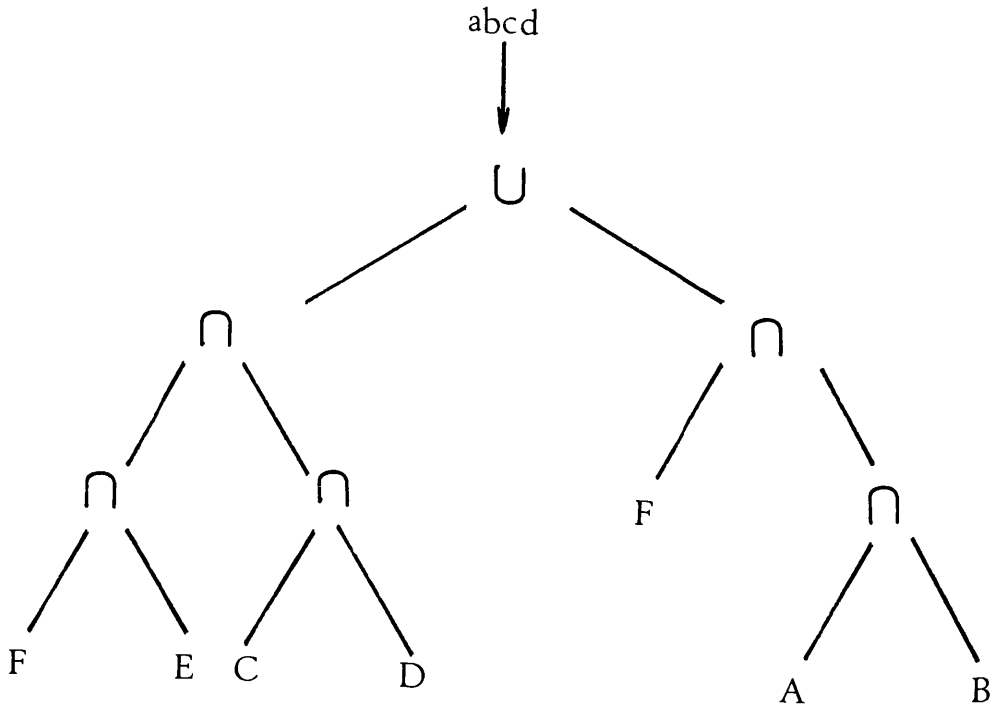
The number of subspaces generated at each stage of the division is usually 2, 4, or 8. If 8 sub-spaces are generated from an even division operation, each of the sub-spaces will have the same shape as the original one (sometimes called their father), which is useful when the sub-divided boxes are organised to form a data structure to hold the model. In the system discussed in this thesis, such an *octree* data structure was used, which will be discussed in greater detail in Chapter 6.

5.2 PRUNING THE SET-THEORETIC MODEL DEFINITION

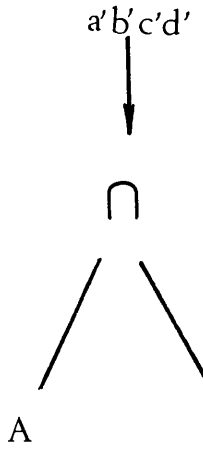
The purpose of subdivision, as was discussed in the last section, is to localise the relationship between half-spaces and each sub-space. However, while the complexity of the relationship will be effectively reduced simply by subdividing the object space in to smaller ones (For example, the intersection between half-space A and sub-space a'b'c'd' in Fig.5.2 can be calculated, which was difficult for the object space abcd), the number of half-spaces related to sub-spaces is not automatically reduced. When evaluating each sub-space, all the related half-spaces have still to be checked, no matter if they lie outside the sub-space or not (For example, the sub-division cannot tell if half-space E contributes to the surface inside sub-space a'b'c'd' or not). To get rid of this unwanted redundant evaluation effort, more sophisticated techniques are required.

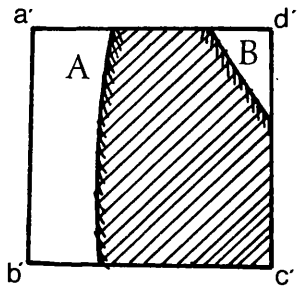
The definition of a model consists of two parts: The set-theoretic expression (an operation tree or CSG tree), and the object space. When the object space is subdivided into subspaces, the operation tree should simultaneously be divided into subtrees corresponding to each of the subspaces, so that each of the subspace will be associated with a subtree that consists of only those half-spaces that contribute to the formation of the particular part of the model inside that subspace. This technique of simplifying the operation tree into subtrees is called *pruning*, because this is rather similar to the pruning of a natural tree (Fig.5.3).

Two kinds of half-spaces are to be removed from the operation tree in order to form a smaller one for a particular subspace. The first is those whose surface lies completely outside the subspace (half-spaces C and D in Fig.5.2 to sub-space a'b'c'd'). In this case, the subspace will contribute to the set-theoretic operation for that half-space either solid or air, and cannot form part of the model's surface that lies inside this subspace. The pruning technique detects such half-spaces and removes them from the operation tree.



(a) The Original CSG Three





(b) A Sub-Tree (c) A Sub-Space

Fig.5.3

The second kind of half-space is those surfaces that, though lying partially or entirely inside the subspace (half-space E in Fig.5.2 to sub-space a'b'c'd'), are excluded by the set-theoretic operations (Fig.5.3). That is to say, though they are spatially related to the subspace, they have simple contributions to the evaluation of the submodel inside this subspace. This is determined by the relationship between half-spaces specified in terms of set-theoretic operations. The result of such simplification can be determined by the contribution of half-spaces to the sub-space and their set-theoretic relationships according to the following two tables for *intersection* and *union* operations.

∩	air	solid	cross
air	air	air	air
solid	air	solid	cross
cross	air	cross	cross

∪	air	solid	cross
air	air	solid	cross
solid	solid	solid	solid
cross	cross	solid	cross

Table 5.1 The Look-up Table for CSG tree Simplification

The pruning operation consists of two basic steps. The first is to check the relationship between a half-space and the subspace to find if the half-space is completely inside, outside or crosses the subspace. If a half-space is outside the sub-space, its contribution to the set-theoretic operation tree is simplified (*air* or *solid* in the above tables); otherwise it is unchanged (marked as *cross* in the above tables). The second is to check whether a surface that is inside or crosses the subspace is excluded by

the operation tree expression. The second step involves the evaluation of the CSG tree according to the above tables. This operation is performed recursively and, in the end, it will eliminate all simplified half-spaces and their *close relatives*, which may or may not be simplified themselves. The result of the operation is a simplified set-theoretic expression, or a *sub-tree*, which corresponds to a particular sub-space and contains only *cross* half-spaces, as show in Fig.5.3. Such a pair of sub-space/sub-tree represents a part of the model. The entire model is represented by a finite number of such pairs, or *sub-sets*. The procedure for generating sub-sets using sub-division and pruning techniques is shown in Fig.5.4. The data structure used to hold such sub-sets will be discussed in the next chapter.

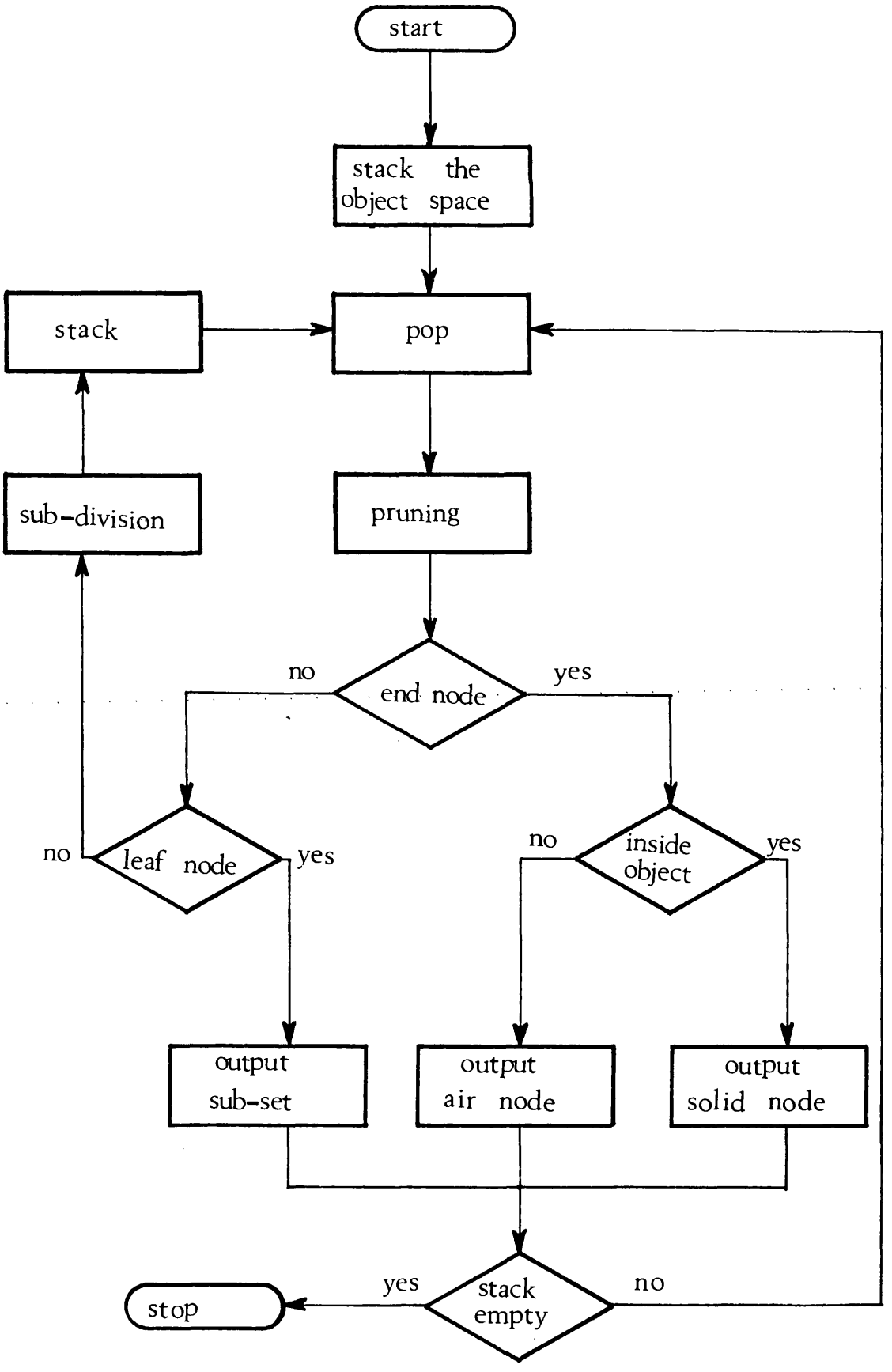


Fig.5.4 Sub-division and Pruning

5.3 THE RELATIONSHIP BETWEEN A HALF-SPACE AND A SUB-SPACE

In order to carry out the pruning operation, the relationship between half-spaces and sub-spaces has to be found. The complexity of the relationship between a subspace and a half-space is affected by the order of the half-space. The simplest one is that between a planar half-space and a subspace. Since a planar half-space defines an infinite plane, it cannot lie completely inside the subspace; it can only be outside it or cross it. The calculation can be done by checking the signs of the planar half-space's potential values at the eight corners of the subspace. The subspace is a *solid* box if all are negative, a *air* box if all are positive, or a *partially solid* box otherwise.

For a curved half-space, the calculation becomes more complicated. The higher the order of the half-space, the harder the calculation. The algorithm for planar half-spaces cannot be used for a curved surface reliably. A curved half-space can be inside, outside or cross the subspace, therefore more cases need to be checked. Furthermore, because of its curvature, a curved half-space's relation to the subspace cannot be determined merely by checking the signs of the potential values at the eight corners of the subspace. This is clearly illustrated in Fig.5.5.

For some simple curved surface types, such as cylinders, spheres or tori, their relationship with the subspace can be obtained by checking the distance between the half-space and the subspace, because the true distance for these types of surfaces can be calculated relatively easily. When this method is used, the subspace is treated as a spherical shape that just bounds the real cubic subspace. The true distance is calculated at the centre of the subspace and then compared with the size of the subspace: If the distance is greater than the radius of the bounding sphere, the half-space is completely outside the subspace and its contribution to the sub-tree is determined by the sign of the potential value at the central point. Otherwise, it is considered to be inside or to cross the subspace. The treatment of a cuboid subspace as a spherical subspace reduces the distance calculations from eight (the eight corners of the box) to one (the centre of

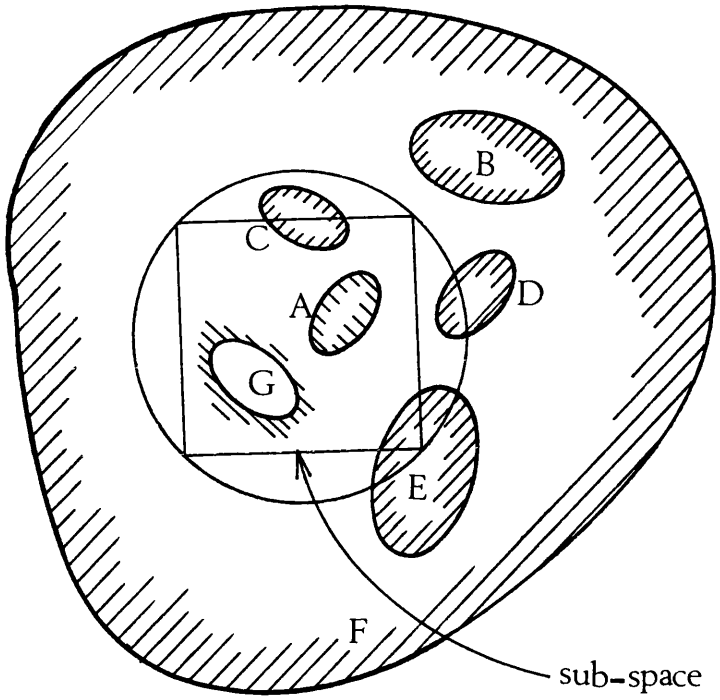


Fig.5.5 Relationship Between Half-Spaces and a Sub-Space

A: Inside the Sub-Space

B: Outside the Sub-Space

C: Crossing the Sub-Space, But Inside the Bounding Sphere

D: Outside the Sub-Space, But Crossing the Bounding Sphere

E: Crossing the Sub-Space and the Bounding Sphere

F: Containing the Sub-Space, But Inside it

the sphere). This obviously saves a great deal of computing time. Though this may treat some half-spaces as crossing the subspace when they are, in fact, not, the result is reliable and greatly preferable to the alternative, which is losing some half-spaces that should be considered as being inside or crossing the subspace, as shown in Fig.5.5.

If planar and simple curved half-spaces are all those used in a solid modelling system, the techniques discussed above are enough to calculate the relationship between them and a subspace. But when more complicated high order half-spaces are used, as in the system presented here, a more general technique is required to ensure the reliability of the calculation.

A general technique for calculating the relationship between a half-space and a sub-space should be able to deal with any kinds of half-spaces. At least, a technique that can deal with a high order half-space should also be able to deal with any lower order half-spaces. For example the technique for simple curved half-spaces can also be used for planar half-spaces.

The difficulty encountered when using high order half-spaces is the calculation of the true distance from a given point. Though the potential value of a point can be easily calculated, it tells little more than the sign of the potential and this information is far from enough to allow the determination of relationship between the half-space and the subspace.

In this solid modeling system described in this thesis, the relationship between a sub-space and a half-space is calculated using the *minimum-maximum test* method, which is discussed in what follows.

Theoretically, the most obvious, reliable and accurate way to find out if a half-space is completely outside a sub-space is to calculate the potential values of all the points that belong to the sub-space. If all of them are positive, the sub-space contains no part of the half-space, if all are negative, it is a solid cuboid, otherwise, it partially

contains the half-space. But in practice, this method would by so time consuming to be impossible to use.

The idea of a minimum-maximum test is to calculate the minimum and maximum potential values of a half-space at all the points belonging to the sub-space by using a small number of calculations and to decide the relationship between the half-space and the sub-space from the calculated minimum and maximum values. Since this does not require exhaustive calculations, it is therefore practically useful.

To illustrate the method, consider the case of a planar half-space, a quadric and a high order half-space (for example, a blend surface).

1. A plane

A plane is the simplest form of a half-space. To obtain the relationship between a plane and a sub-space using the minimum-maximum test, first we calculate the potential value at the central point of the sub-space. Since the potential value of a normalised plane (such as those used in this system) represents true distance, by shifting the potential by the size of the sub-space in both the positive and the negative direction from the plane, we can obtain the possible minimum and maximum potential values. (This can be simply the radius of the bounding sphere of the sub-space, or more complicatedly the real dimension of the sub-space in relation to the plane.) Checking the signs of the results, we can then easily determine the relationship between the plane and the sub-space: If both are positive, the sub-space is outside the planar half-space, if negative, it is inside; otherwise it crosses the plane, as show in Fig.5.6.

2. A Quadric

Because different equations can represent the same quadric shape [Chapter 3], the potential value of a point may or may not represent the true distance between the point and the quadric. Taking a cylinder as an example, equations (3.7), (3.11) and (3.16b) [Chapter 3] can all be used to represent the same cylindrical surface. If equation (3.16b)

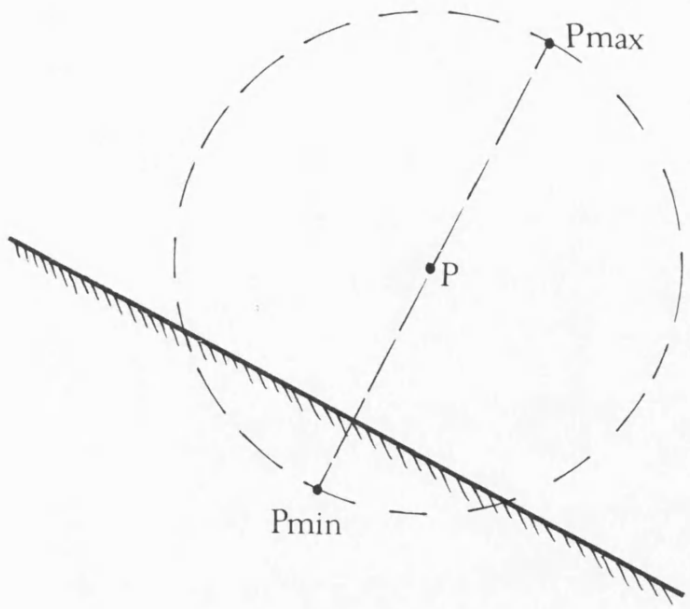


Fig.5.6 Minimum-Maximum Test of a Plane

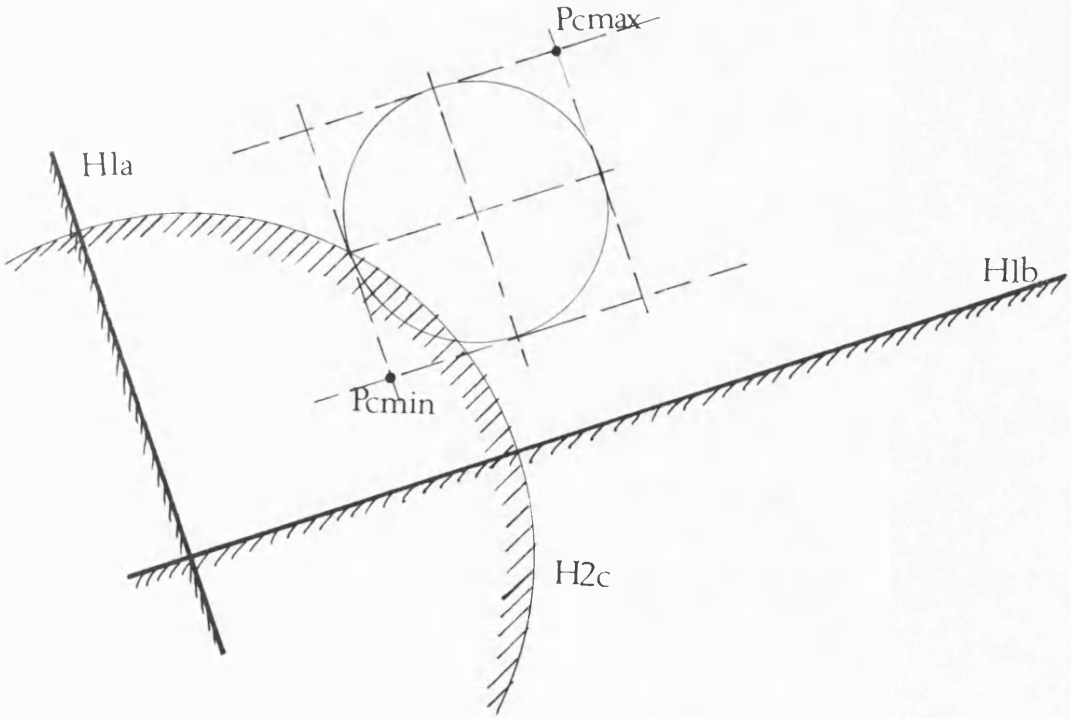


Fig.5.7 Minimum-Maximum Test of a Sphere

is used, the potential value at a point will give the true distance between that point and the cylinder; if the other two are used, the potential value will not represent true distance. For example, equation (3.7) will give a point 5 units to the inside of the surface a potential value -75, while giving a point 5 units to the outside of the surface a potential value 125. If the radius of the cylinder changes, these potential values will also change, even if the distance is still 5 units from the surface. Therefore we cannot reliably obtain the minimum-maximum values just by shifting the potential by the size of the sub-space.

Since planar half-spaces are the basic elements used in constructing a curved half-space (the polynomials are formed by multiplying plane equations together [Chapter 3]), the minimum and maximum values of a quadric can be calculated by making use of the minimum and maximum values of its defining planar half-spaces. The following are the steps for calculating the minimum and maximum values of a quadric surface.

Firstly, the possible minimum and maximum values for each of the planar half-spaces which make up the equation of the quadric are calculated using the method described earlier.

Secondly, we use all these values calculated above in the calculation of the potential values of the quadric.

In evaluation of the expression for calculating the minimum and maximum values, each operand will have two values, one minimum and the other maximum. If an operand is a constant, the minimum and maximum values are the same; if it is a planar half-space, the minimum and maximum values are determined using the method described earlier. For a mathematical operation on two operands, their minimum and maximum values are all used in a combination, so that the results will be the possible minimum and maximum values from those of the two operands. The calculated minimum and maximum values are then used in subsequent evaluations,

until the evaluation of the expression of the quadric finishes. The results will be the minimum and maximum values for that quadric surface.

Finally, by checking the minimum and maximum values, it is then possible to decide the relationship of the sphere and the sub-space. Fig.5.7 shows an example for calculating the minimum and maximum values for a sphere.

3. A High Order Surface

For high order surfaces or blend surfaces, the minimum and maximum values are obtained in the exactly same way as for a simple curved half-space. The only difference is that this is performed recursively until the final stage is reached.

Because of the non-linear property of high order half-spaces, the calculated minimum-maximum values are less representative than that for low degree half-spaces. In other words, sometimes a half-space that is actually completely outside a sub-space may be considered (according to the maximum and minimum results calculated by the program) to cross the sub-space. Fortunately, this is always on the safe side: sometimes the program may fail to eliminate all the redundant half-spaces on a sub-tree, but no half-space that is really related to the sub-space is lost. This may in practice create some false non-empty boxes, but they do no harm to the modelling of the object. The false non-empty boxes are often eliminated at a further stage in the division, when the relationship between the surface and sub-spaces become simpler due to the reduction of the sub-space size, as we discussion earlier.

For simplicity of discussion, the above used only minimum and maximum values in calculating minimum and maximum values for curved surfaces. However, in practice, a middle point is also used in the minimum and maximum value calculations for curved surfaces. The main reason for using a middle point in addition of the minimum and maximum values is that, when a plane crosses the sub-space, the minimum and maximum values calculated using the above method represent the

minimum and maximum potential values, but not the absolute distance, as shown in Fig.5.8. If only minimum and maximum values are used, the result would be unreliable. The use of a middle point increases the amount of evaluations, but it makes the calculation reliable. The value of a middle point is decided as follows: if the minimum and maximum values have the same sign, the middle point will be their average value, otherwise, the middle point will have the value 0 (because, in this case, the surface lies between the minimum and maximum point, therefore, the minimum absolute distance must be 0).

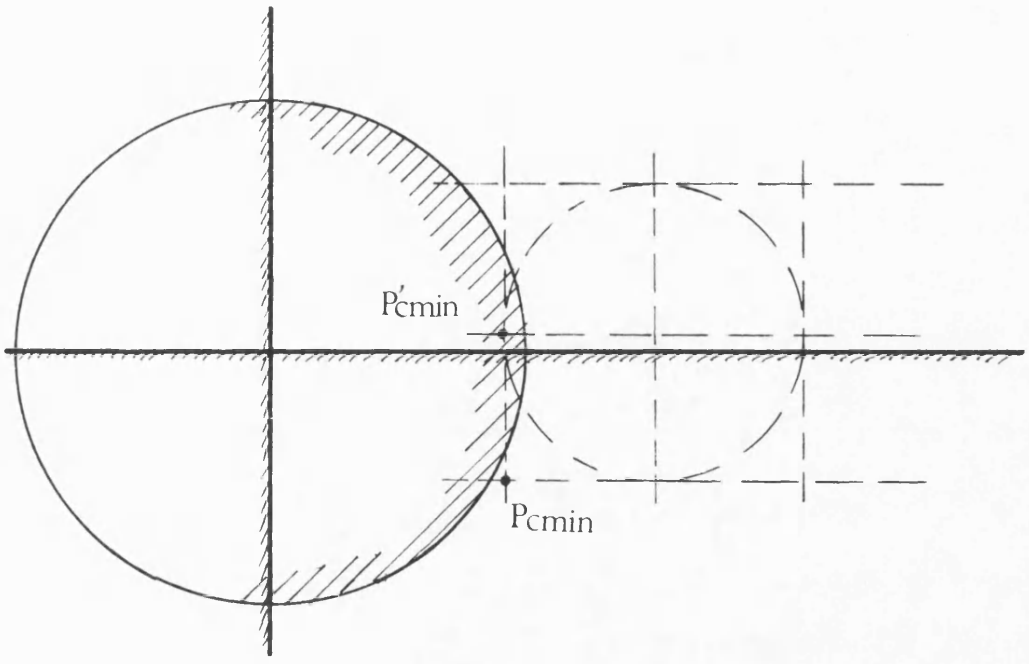


Fig.5.8 Use of Middle Points in Minimum-Maximum Tests

CHAPTER 6

THE DIVIDED AND PRUNED MODEL

The most important feature of a solid modelling system, as was discussed in Chapter 2, is the data structure used for the internal representation of the object. In this system, a data structure called an *octree sub-set model* is used to hold the geometric and other information about the solid model.

6.1 BASIC REQUIREMENTS OF A SOLID MODEL DATA STRUCTURE

A solid model can be held in many different data structures, such as a Boundary Representation, or a CSG tree. Within each category, many different kinds of actual data arrangements can be used, and almost all commercial and research systems differ in this respect. It is hard to say which data structure is better than the others. However, some requirements are basic to all of them. According to these basic requirements, one data structure can be compared to another. Whatever data structure is used, it should meet these requirements to be satisfactory in its applications. Some of the basic requirements are as follows.

a. Uniqueness

The data structure must represent a unique model. A solid model which is ambiguous in geometry can hardly be used for any serious engineering application.

b. Completeness

Information completeness is another basic requirement for a solid model. Apart from geometric information, which is the most important, the model may also include information about material, surface finishing, tolerances etc. In many cases, it is not necessary to have all of this information, or even all of the geometric information. For example, a model of an assembly for display purpose may ignore internal parts or those trivial parts that are too small to be seen on a graphics screen. But a model with more geometric information is certainly more versatile.

c. Accuracy

Accuracy is important to all solid models, especially to models of engineering components which are to be produced automatically on NC machines using the geometric information held about the model. Models of robotic motion are obviously also required to have accurate geometric information. Solid models which are

approximation for such purposes must be constructed carefully so as not to cause any serious inaccuracy in their applications.

d. Compactness

Though not a crucial requirement, it is important to make the best use of the computer's resources. A more compact data structure means that more, larger and more complicated objects or assemblies can be modelled using the same equipment and other resources.

e. Readiness

This requirement is very much application dependent. The purpose of a solid model is for applications in design and manufacturing and other activities. It takes time to generate the data required for pictures, analysis, finite element meshes or NC instructions. The closer the data of the model is to the data needed by the applications, the quicker the computation will be.

6.2 MODEL DATA STRUCTURE

The results of the division and pruning process are a number of subspaces and their corresponding subtrees of set-theoretic expressions. For each pair of the sub-space/sub-tree set, or *sub-set*, the relationship between the subtree and the subspace is relatively simple: each subspace usually contains 0, 1 or 2 half-spaces, a few contain 3 half-spaces, and in very few cases, the number of half-spaces is greater than 3. This is from an original object space containing, possibly, thousands of half-spaces.

From these subsets, several types of data structure can be generated to hold information about the modelled object. They can be subsequently used as independent data in various applications, such as mass property calculation and finite element analysis. The data structure is introduced here.

6.2.1 Cell Model

A cell model is a pure spatial representation of an object. If the size of a sub-space is small enough, the geometric structure within it can be ignored. Each sub-space can be viewed as either completely empty or completely solid. There are several ways to decide if a sub-space should be air or solid. One simple way to do this is to test the central point of the sub-space, using a membership test [Chapter 7]. If it is inside the object, the sub-space is considered to be solid, otherwise, it is considered to be air.

A cell model represents a unique solid model. However, as the geometric information is simplified, it becomes less complete. For example, the boundary information can no longer be extracted because the definition information for each half-space has been lost. The accuracy of a cell representation is controlled by the size of the cells. Theoretically the model can be as close to the real object as the user wishes, but in practice, this is limited by the storage available on, and the computing speed of, the computer.

In applications, a cell model is good enough for some numerical analyses, such as for calculating volume, weight, centre of gravity, and so on. It can easily be used in modelling objects such as buildings and normally it is simple and quick to manipulate. But a cell model is not good for applications where accurate surface information is required, such as shaded picture generation or for producing NC instructions to cut object surfaces.

6.2.2 Polygon Model

As was just discussed, the disadvantages of a cell model are its lack of surface information and the fact that the size of each sub-space has to be very small. To improve these aspects, a polygon model can be used instead.

From each sub-set resulting from the recursive division and pruning process, one or more polygons can be generated to represent the surface structure inside the sub-space. If the size of the sub-space is reasonably small (It could, however, be much bigger than that of the cell model), the polygon(s) can be good enough to represent the original surface for many purposes (picture generation, for example).

To generate a polygon model, only partially occupied sub-spaces are evaluated. If the size of a subspace is small enough for us to assume that no half-space is completely inside this subspace, and that the half-spaces crossing this subspace can be considered to be planar surfaces in relation to this subspace, then each related half-space will intersect the sub-space at some of it 12 edges. As a plane can cut a subspace at one of its corners, or does not cut the subspace at all (if it is only *set-theoretically* related to the subspace), the number of intersection can be from 0 to 6, as illustrated in Fig 6.1. If the number of intersections is less than 3, it effectively represents no surface and hence is ignored. Otherwise, the intersections are sorted into a clockwise or anticlockwise sequence to form the vertices of a polygon. This polygon approximately represents that particular part of the object surface contained in the sub-space.

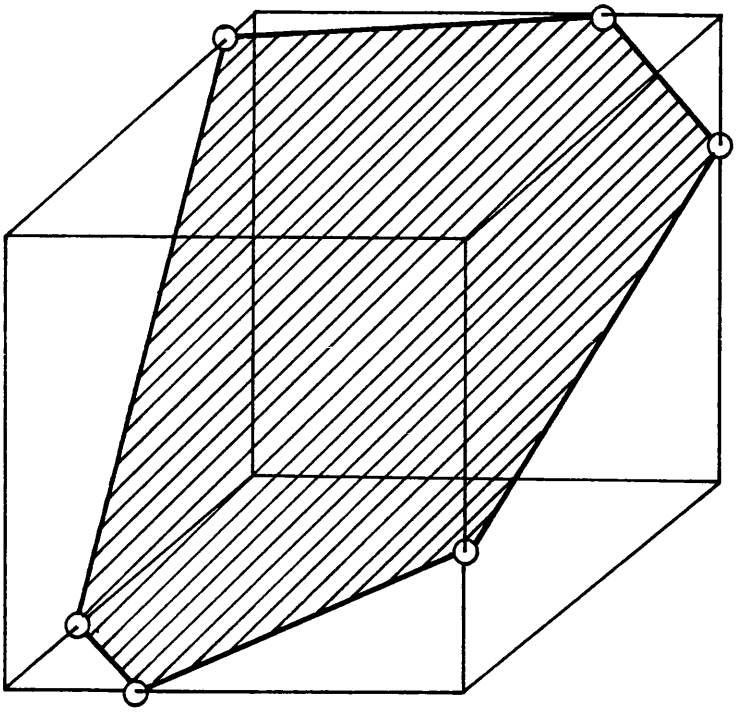


Fig.6.1 Intersection of a Plane and a Sub-Space

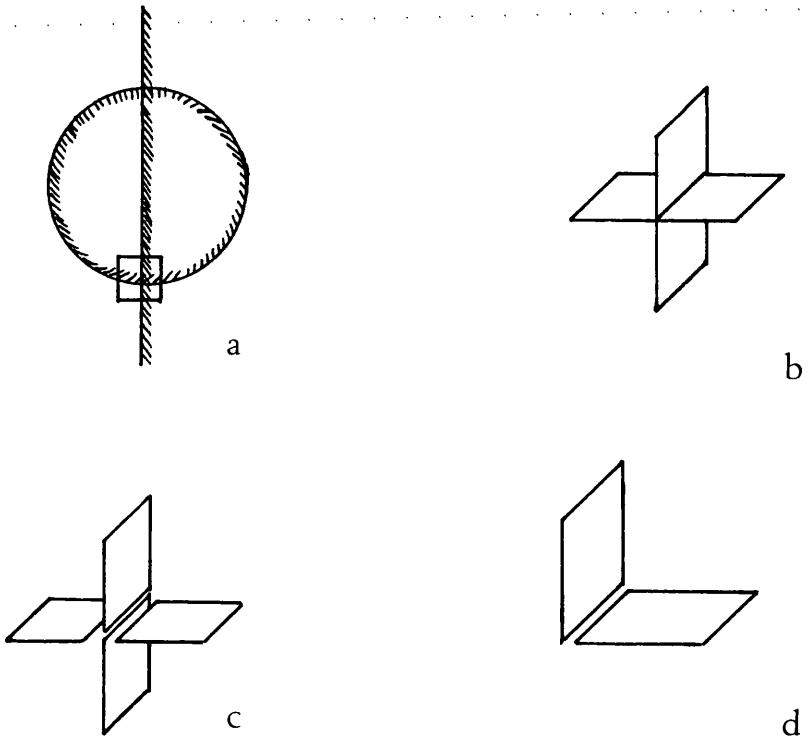


Fig.6.2 Steps for Polygon Generation

However, the object's surface may be constructed from more than one half-space in the sub-space. At intersections of two or more half-spaces, the situation becomes more complicated. This complexity is detectable from the length of the set-theoretic expression sub-tree. No matter how small the sub-space size may be, this complexity will remain as long as the sub-tree contains more than one half-space.

If the model is just for producing pictures, Warnock's algorithm can be used to divide the sub-space into even smaller ones until it contains only one half-space, then the above method can be used to generate a single polygon; or until its size is too small to the resolution of the display device, then it can be thrown away without degrading the picture. But if the model is also for other purposes (such as geometric analysis and triangulation), this difficulty has to be dealt with more carefully.

In this system, a sub-set that contains more than one half-space was evaluated using the following algorithm to generate polygons (Fig.6.2).

a. Consider each half-space separately and generate a polygon for each of them by calculating intersections between them and the sub-space using the method described above.

b. Chop each generated polygon along its intersection lines with other half-spaces belonging to the same sub-set. This will increase the number of polygons and reduce their size. If two polygons do not intersect each other, no chopping needs to be done between them.

c. Perform a membership test [Chapter 7] for each of the polygons, keeping those that are part of the surface and throwing away the others. The resultant polygons are then stored as a number of vertices, along with their normal vectors if required. These normal vectors can be calculated from the vertices and their sequential direction (clockwise or anticlockwise), or can be calculated from the gradient of the half-space by differentiating at the middle point of each of the polygons.

A polygon model is unique in the sense that once it has been created, it will represent an unambiguous solid object. However the formation of polygons for a given model is not unique. It depends on how the object is placed in the object space, how the object space is divided and how small the sub-spaces are. If the object is rotated through an angle, the shape and size of the polygon(s) inside the sub-spaces will change, but as long as the sub-spaces are small enough, this change will not affect the level of approximation to the real object.

Since the original curved half-spaces are approximated by polygons, the geometric information is not complete and accurate. But a polygon model can be more readily used for some applications, such as picture generation (Fig.7.4, 7.5). As the vertices of these polygons are exactly on their original half-spaces, a polygon model can be used in applications where accurate data are required, if the size of the polygons are small enough. For example, a well generated polygon model can be used for NC machining instruction generation quite satisfactorily.

6.2.3 Faceted Model

A faceted model is, in fact, a simplification of a polygon model. A polygon model is stored as a number of vertices and their normal vectors. If the number of polygons is large, the storage needed would be very large. The idea of a faceted model is to reduce the amount of data needed for the model. For example, to store a polygon with 4 vertices and the normal vector, at least 15 values have to be stored; furthermore, as each polygon is represented and stored separately, common vertices between adjacent polygons have to be stored twice or more. If a polygon can be represented by a plane, only 4 values need to be stored. As we discussed earlier [Chapter 3], these 4 values are the coefficients of the plane equation, which are sufficient to define the position and orientation of the plane. The direction of the plane will be the same as the normal vector of the polygon and the distance between the plane and the origin can be calculated from the middle point of the polygon.

Since the vertices of a polygon are not generally co-planar, because they are intersections of curved half-spaces with straight lines, the plane generated from the polygon is not co-planar with all the vertices of the polygon.

The simplification of the data form from polygons to planes reduces the storage requirement a great deal, and hence speeds up some of the subsequent calculations.

A faceted model can also be constructed from a real object. For example, the data obtained from measuring an engineering component using a coordinate measuring machine are distances and directions, from which the co-ordinates of the measured points can be calculated. The co-ordinates in turn can be used to define planes by grouping adjacent points together to form a triangle. This is useful if the model inside the computer is to be compared with the real object (currently research is being done on this subject at Bath university[113]).

6.3 THE OCTREE SUB-SET MODEL

A common shortcoming of all the models discussed above is that the original geometric and set theoretic information is simplified, changed or even completely lost. Though in many cases the results can meet the requirements of applications, they are limited by their inaccuracy and can hardly be considered to be a robust, reliable and flexible solid model. Also the range of applications they can be applied to is rather narrow. A data structure combining a sub-set model and an octree is used to improve the weak points of the other data structures discused earlier.

6.3.1 Sub-Set Model

A sub-set model does not change or modify the original geometric and set theoretic information, but the data structure is organised more efficiently. Each sub-set is stored as a data unit. A cluster of sub-sets together form the complete model of the object.

In a polygon model or faceted model, the content of a sub-space is different from any other sub-space and the polygons or planes have to be stored separately, while in a sub-set model, since many sub-spaces may have the exactly same sub-tree (particularly for sub-spaces close to each other), it is not necessary to store the content of each sub-space separately. For the entire model, in comparison to a polygon or faceted model, only a very small number of different sub-trees have to be explicitly stored, all the sub-spaces that have the same sub-tree can be related to that sub-tree by a simple pointer.

In comparison to other model data structures, a sub-tree model is much more compact, accurate and complete. It is also unique. Though it is generally less easy to use, it can be applied to many more different problems.

The main idea of a sub-set model is to construct a more flexible model that has

complete and accurate geometric information. In this system, the sub-set model is held in an octree data structure, so it is called an *octree sub-set model.*

6.3.2 The Octree

An octree is the equivalent of a binary tree (Fig.6.3) in one dimensional space and a quad tree (Fig.6.4) in 2 dimensional space [103]. It is a natural data structure for three dimensional models (Fig.6.5).

Though other data structures, such as a binary tree applied to three-dimensional space [83], can used to hold a CSG model, an octree is more widely used for three-dimensional representations. It is consistent, complete and unambiguous. Usually, an octree is constructed by symmetric recursive indexing, that is, each space is divided into eight subspaces, or octants, of identical shape and volume. This division continues until either a subspace contains no surface, or reaches the smallest size defined by the user or limited by the device used.

In practice, an octree can be formed in different ways. For example, it can be built up with tiny elements, which are generated by other techniques [36], or it can decompose a single entity [57], such as the object space described in this thesis.

An octree is only a data structure used as a framework for holding geometric information, the quality of the data held in an octree depends on the data type of the octree's leaf nodes (the smallest volume of the octree). A leaf node can be air, solid or partially occupied. For an air or solid leaf node, its form is the same in every octree type and is labeled by an indicator at the leaf node position to show if it air or solid.

All the model data types described earlier can be used as the leaf node data types of an octree. If cells from a cell decomposition representation are used as the node data type of the octree, a leaf node can only be either solid or air. All the solid leaf nodes are identical, and so are air nodes. However, if other data types, such as polygons or facets, are used, a leaf node may also be partially occupied, and would thus be different from

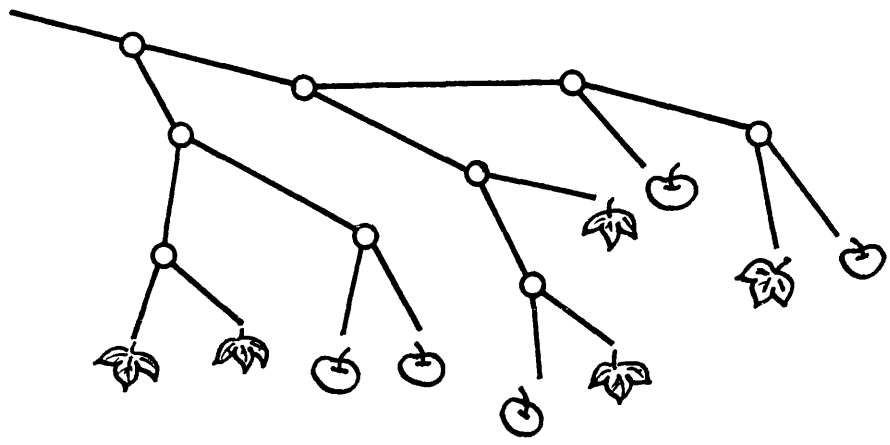


Fig.6.3 A Binary Tree

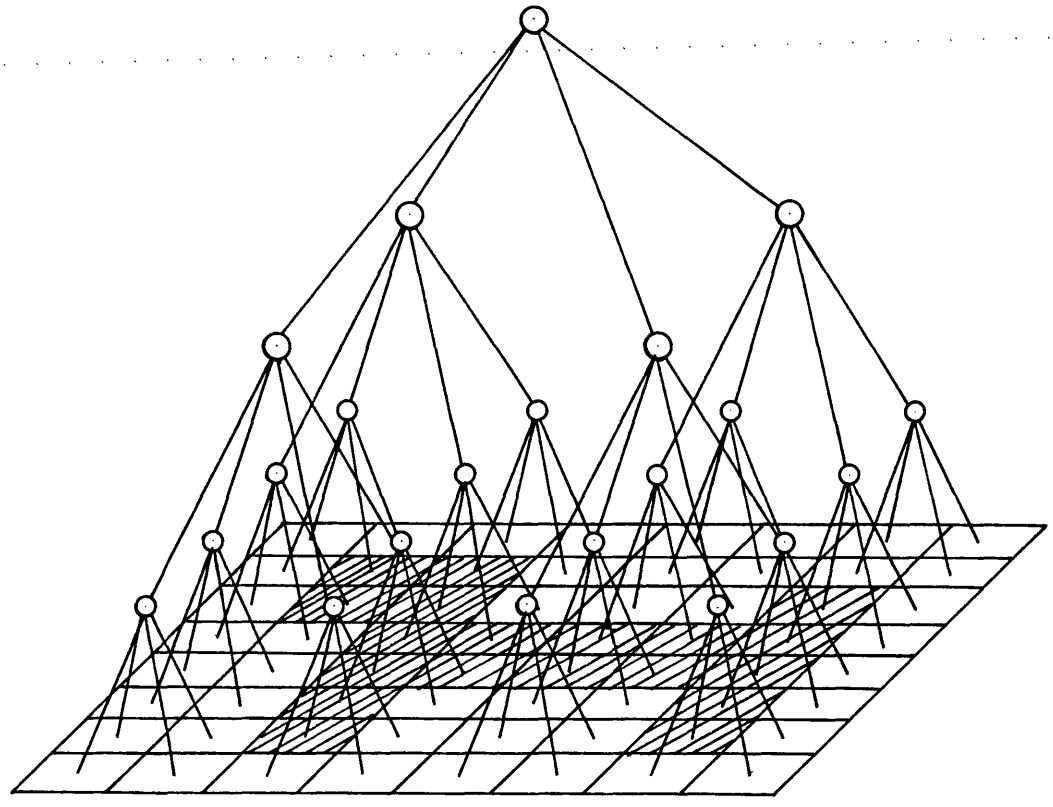


Fig.6.4 A Quad Tree

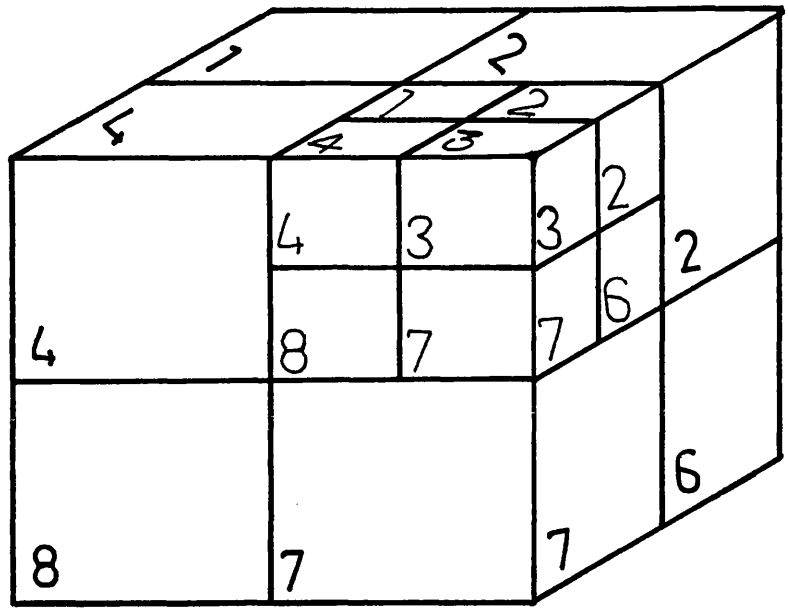


Fig.6.5 An Octree by Subdivision

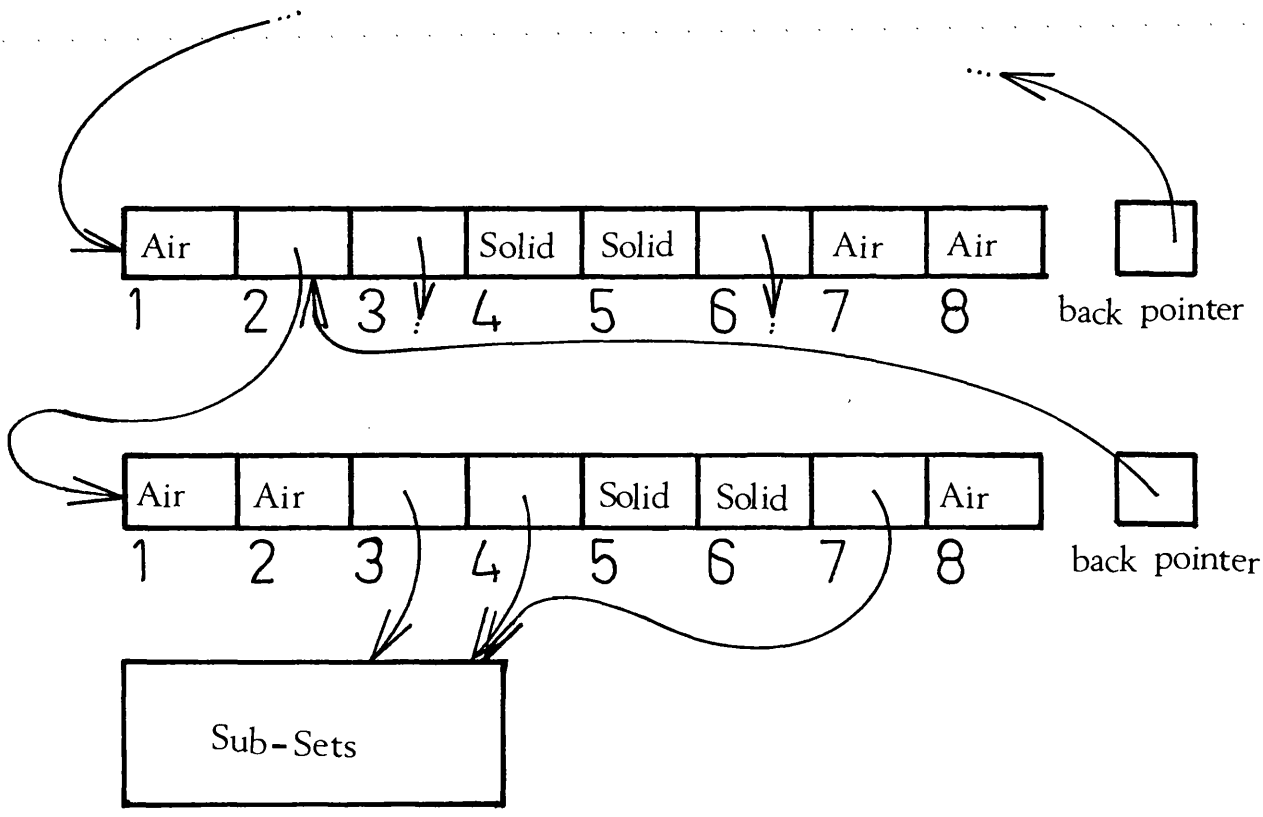


Fig.6.6 Data Structure of the Octree

one to another.

The partial node type is used to improve the precision and conciseness of the octree representation. The type of a leaf node determines if the representation is exact or approximate, the size of storage needed and speed of access to and manipulation of the data.

It is not easy to make an octree good in all respects. An advantage in one aspect is often associated with a disadvantage in another. For example, the cell representation is quicker to manipulate (for example, for mass calculation) than the subset representation, but is less precise and concise. Therefore, one octree can be very different in its behaviour from another, if they have different leaf node types. Which type is to be used should be determined by the application and its requirements.

6.3.3 The Construction of an Octree Sub-set Model

In this system, the division begins at the object space. The first eight sub-spaces divided from it are called the first generation, which can be further divided up to form another 64 sub-spaces, which in turn are called the second generation. Each stage of division creates a new generation as shown in Fig.6.5.

Each sub-space at each different division stage is called a node. The biggest one, the object space, is called the root; the smallest ones, at which the division stops, are called leaf nodes or leaves; those nodes having zero sub-tree length are called end nodes. These may or may not be the smallest sub-spaces. If a node has eight sub-spaces, it is also known as the father of the eight sub-spaces, which are called the eight sons of that node.

A node is related upwards by a back-pointer with its father (except for the root) and downwards with its eight sons (except for the leaf nodes and end nodes). The eight sons of a father node are numbered from 1 to 8 according to their position relative to their father, as shown in Fig.6.5.

The decision as to whether a node should be further divided is made by the sub-division and pruning operation. If the length of the sub-tree of a node after pruning is 0, no further division is required and this node is flagged as an end node. If the result of the pruning operation shows that the node is inside the object, it is marked as a solid node; otherwise, it is marked as an air node. If the length of the sub-tree is non-zero, the division is carried out further until reaching either an end node or a leaf node.

The octree data structure is stored in a two-dimensional array. Each row contains eight nodes of the same family, and a back-pointer to their father's address. A integer value is given to each node according to its status. If it is an end node, the integer indicates if it is solid or air; if it is non-leaf node and not an end node, the integer indicates the address of its eight sons; if it is a non-end leaf node, the integer indicates the address of the corresponding sub-tree, as illustrated in Fig.6.6.

The use of an octree data structure makes it easy and quick to locate a particular sub-set. For example, given a point in space, the position and content of a sub-set which the point belongs to can be easily located. This is very useful in ray casting for picture generation, for example, as it is necessary to find all the leaf nodes along the ray†. A top down search can be used to find the right node, by comparing the position of the point to the middle point of a node to determine which son it belongs to. This is carried out recursively until reaching an end node or a leaf node.

6.3.4 Features of The Octree Sub-set Data Structure

The octree sub-set data structure has several advantages over the other data structures. The main features of this data structure are as follows.

a. It is organised in the most concise way. That is to say, no node contains eight solid sons or eight air sons. It is of the minimum size conditional on the smallest leaf node size.

† The ray is chopped up into small segments, which are checked one by one against the octree until the ray hits the object.

b. The use of the subset as the leaf node type makes it concise, precise and informationally complete. Also, it requires less storage for the half-spaces, because many leaf nodes are often associated with the same subset, unlike the cases of polygon or faceted models where each leaf node must have a different geometric information content.

c. It solves the problem of ambiguous relationships between half-spaces that are in the same subspace. This is often the case in faceted representation, because no information about the se-theoretic definition of the subspace is kept for them.

d. It is more versatile and can be used for more applications and behaves more robustly. Also, the other representations, such as cell, polygon and planner faceted models, can be derived from this data form.

CHAPTER 7

PICTURE GENERATION

Computer graphics are one of the main branches of computer technology. It is computer graphics that has made it practical to use computers in design and manufacturing activities and which has formed the basis for almost all CAD/CAM systems.

Picture generation is one of the fundamental requirements for all solid modelling systems. Because of their crucial importance in the design, analysis, manufacturing, communication and documentation of engineering components, pictures of a solid model must be strictly consistent with the modeller's internal representation. This requirement makes them different from other computer graphics images (such as those used for computer film animation, for instance) and more difficult to produce.

7.1 COMPUTER GRAPHICS ENVIRONMENT

7.1.1 Importance of Computer Graphics

Techniques of computer graphics have been developing rapidly, different kinds of graphic devices have been invented and produced, and a great number of remarkable pictures have been generated. Computer graphics has found its applications in various areas, such as drawing, simulation, film making and TV advertisement production.

A computer equipped with graphics facilities can be used to do many things. It can be used for mathematical research to help in the viewing of various curves, surfaces and other images of many abstract functions and expressions that are too difficult or too complicated for the mind to grasp correctly and accurately. It can also extend the dimension of human sight; for example, a four dimensional shape can be projected to three-dimensional space. Generally speaking, computer graphics technology can potentially enable almost anything to be brought into the human scale to be looked at. With different kinds of graphics devices, different kinds of image can be produced, modified, represented, stored and retrieved.

If a solid modelling system is used as the basis for generating computer graphics, capabilities for design and manufacturing can be greatly extended and improved. A design can be viewed, analysed, tested and modified quickly and accurately. The product can be seen and appreciated before it is actually produced. This will of course facilitate this work a great deal.

7.1.2 Display Devices

Several MX-1024X/8/8/32 Colour Graphics Generators, produced by Mirconex Limited, have been used to generate pictures from this solid modelling system. They were assembled from eight MX-QD2 single-plane generator boards having a 1024 x 1024 image store and a 1024 x 512 pixel viewing window (allowing two pages of 1024

x 512); 256 colours were allowed on the screen at any one time from a palette of 16.7 million colours. Digivision and CDCT3/51 monitors were used to display the various pictures from these graphics generators.

Hard copies of pictures on the screen could be taken in two ways. One was to use a slidemaking camera machine to record the picture on film from the video input to the monitors, which could then be developed either as slides or printed on photographic paper, as the shaded pictures show in this thesis. The other hardcopy method was to make a monochromatic hard copy directly on a silver-halide hard copy machine which was connected to the display device. Though colour could not be recored on such hard copies, they were much quicker to produce.

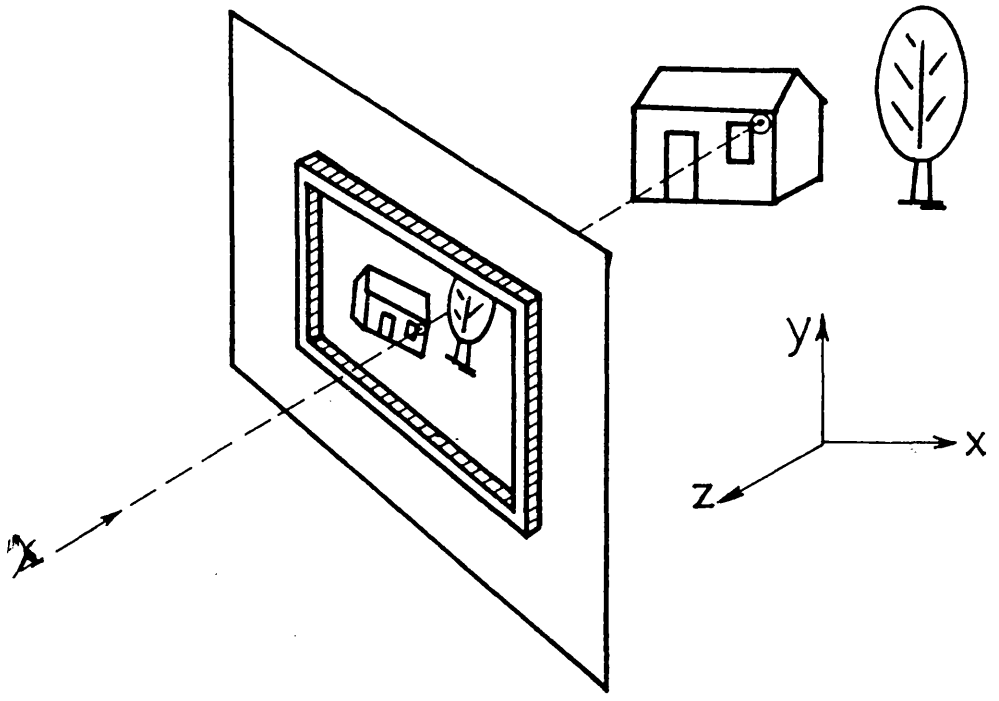
7.1.3 Picture Projection

There are usually two ways to record the image of a three-dimensional object on the two-dimensional plane; they are *parallel projection* and *perspective projection*. For either of them, the geometric data in *world coordinates* have to be transformed to the *screen coordinates* to form the right image, as show in Fig.7.1.

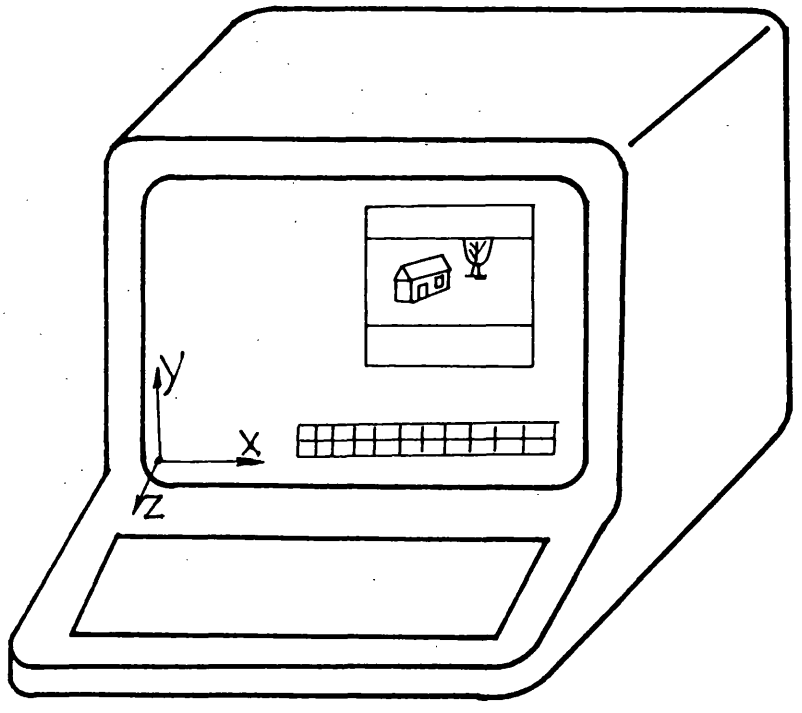
Pictures made by projecting the scene onto the *viewing plane* in a constant direction parallel to the *viewing vector* are called parallel projections. On such a picture, only depth perpendicular to the *viewing plane* was changed (ignored in fact), while others are recorded exactly as they were in the world coordinates (Fig.7.2c).

Pictures of perspective projection are generated by viewing the object in a similar way as a camera taking a picture. On such a pictures the dimensions of the object change along the viewing direction and hence they show the effect of distance between the object and the viewer (Fig.7.2d).

In the system described here, pictures were generated by parallel projection. The advantage of using parallel projection was that the distances perpendicular to the



(a) transforming a image onto the screen



(b) screen coordinates and viewport

fig. 7.1

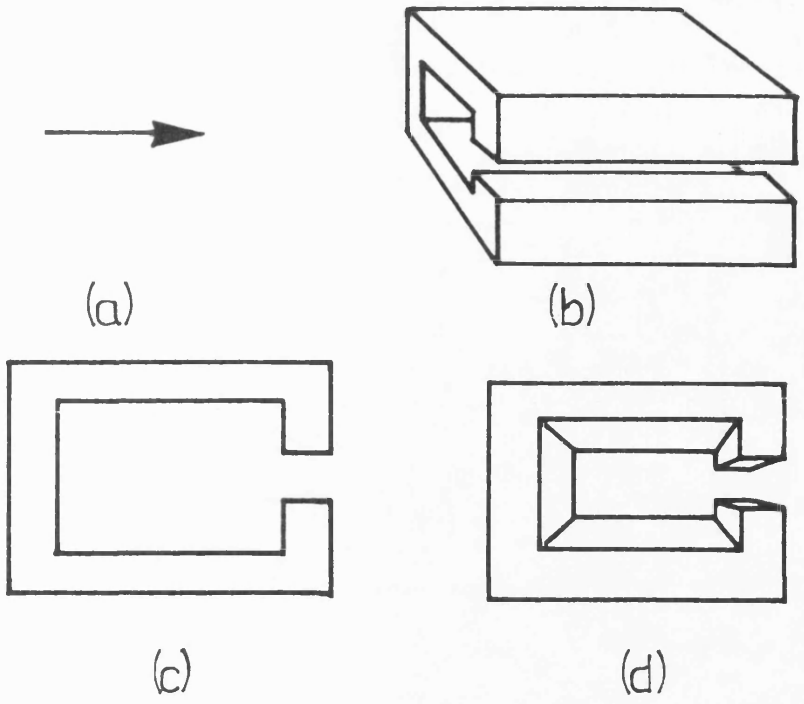


fig.7.2 picture projection

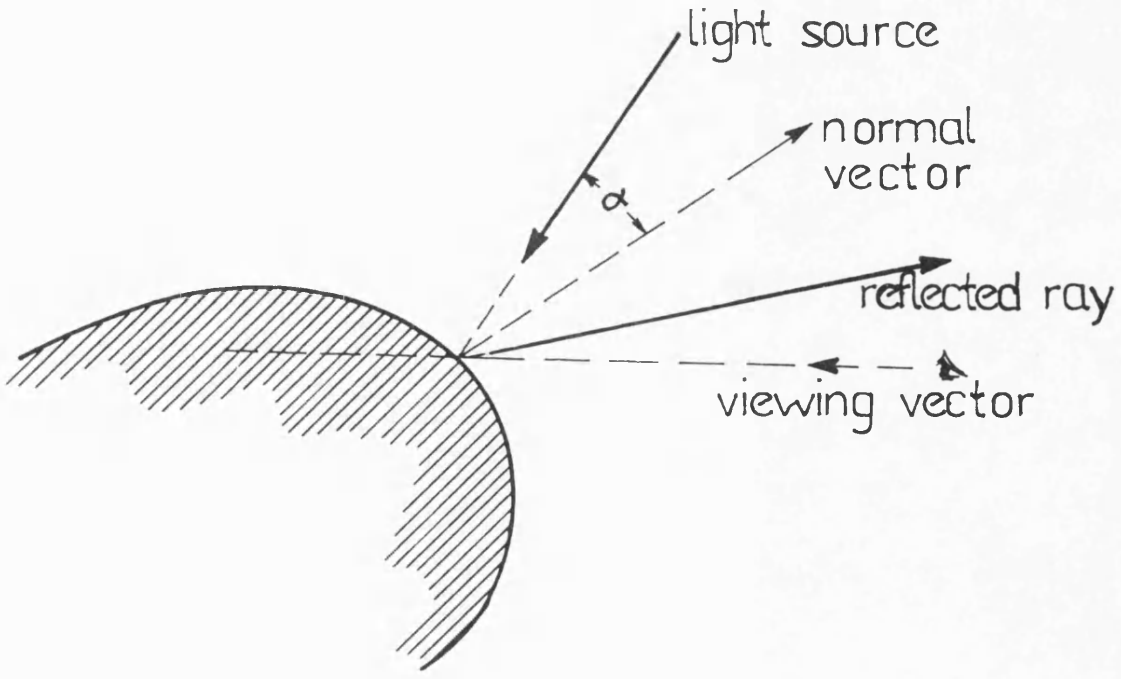


fig.7.3 shade calculation

viewing direction were truly recorded. This was very useful in combining images of an object with the NC codes generated for producing it [Chapter 8].

7.1.4 Colouring and Shading

In a shaded picture, different points on the object should be assinged different colours and intensities. Using Lambert's cosine law, we can calculate the intensity I of a point under a single light source as follows

$$I = I_s K_c \cos\alpha + I_s A_m \tag{7.1}$$

where I_s is the intensity of the light source, α is the angle between the direction of the light source and the normal vector of the surface point, K_c is the reflectance coefficient of the surface, and A_m is the ambient light. If α is equal to or greater than 90 degrees, the surface at that point is facing away from the light source and no light is reflected, hence the intensity of the point is equal to the degree of the ambient light.

In the presented system, a colour band is assigned to a solid model or part of a model (this is useful in dividing the object's surface into several regions for more efficient NC machining [Chapter 8]). A colour band has a certain number of shades based on a particular colour. For a single picture, a total number of 256 shades can be used. A colour band can be a portion of them; for example, a colour band of 30 shades can placed between 71 to 100 or 141 to 170. Usually several colour bands are used for a picture to show different parts of the object or different kinds of surfaces. If a model is given a colour band of $2C$ shades starting from C_s, then the actually shade number S for a surface point can be calculated as

$$S = C_s + (2C\cos\alpha + C) \tag{7.2}$$

where α is the angle, which can be of any value from 0 to 180 degrees, between the surface normal vector at the point and the light source (Fig.7.3).

To model an object more faithfully, or if special effects are to be achieved, several other factors need to be considered. Techniques of lighting with more light sources, highlighting, shadowing, mirrors, transparency and texture are all important to model an object more realistically[22][23][25][26][33][70][71][73].

7.2 TYPE AND QUALITY OF PICTURES

Different types of pictures could be generated from a solid model. The following are some basic types of pictures that can be generated from the author's solid modeller.

7.2.1 Section Views

A section view is a line drawing picture that represent the outline of the model in a particular plane. This plane can be placed in any position and orientation in space, enabling the user to view the object from different angles and depths. The advantage of a section view is that it is much quicker to produce than a shaded picture. Therefore, it is suitable for design and modification interactively (Fig.4.7).

7.2.2 Polygon Faceted Pictures

A shaded picture can be made up from a number of polygonal facets. If the size of each polygon is reasonably small, it can be considered to be close enough to the actual surface of the object, hence the picture will be good enough to represent the shape of the real object.

To create shades for the picture, equation (7.2) can be used to calculate a shade number for each polygon. Only one shade is given to a polygon, which reveals the effect of the approximation. The smaller the size of the polygon, the closer the picture is to the real shape and the smoother the shading is.

There are several ways to create a picture from polygon data source or other data source, [21][24][29], But in this system described in this thesis, the painter's algorithm [64] was used to draw polygons onto the screen from back to front, the result being a shaded picture with the hidden surfaces removed, as shown in Fig.7.4 and Fig.7.5.

7.2.3 Wire-Frame Pictures

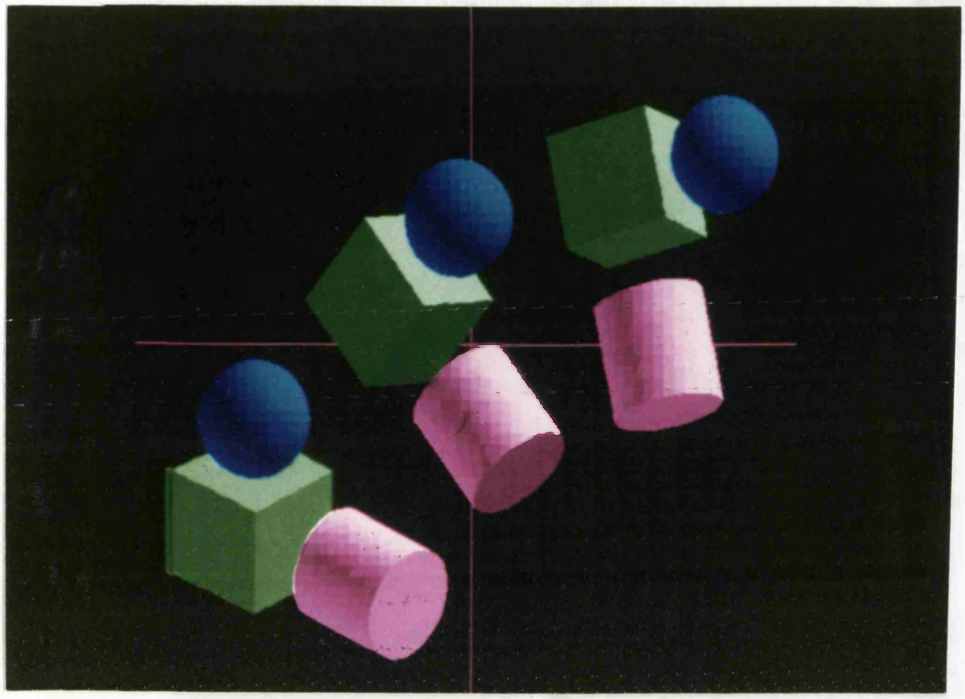


Fig.7.4 Polygon Models (1)

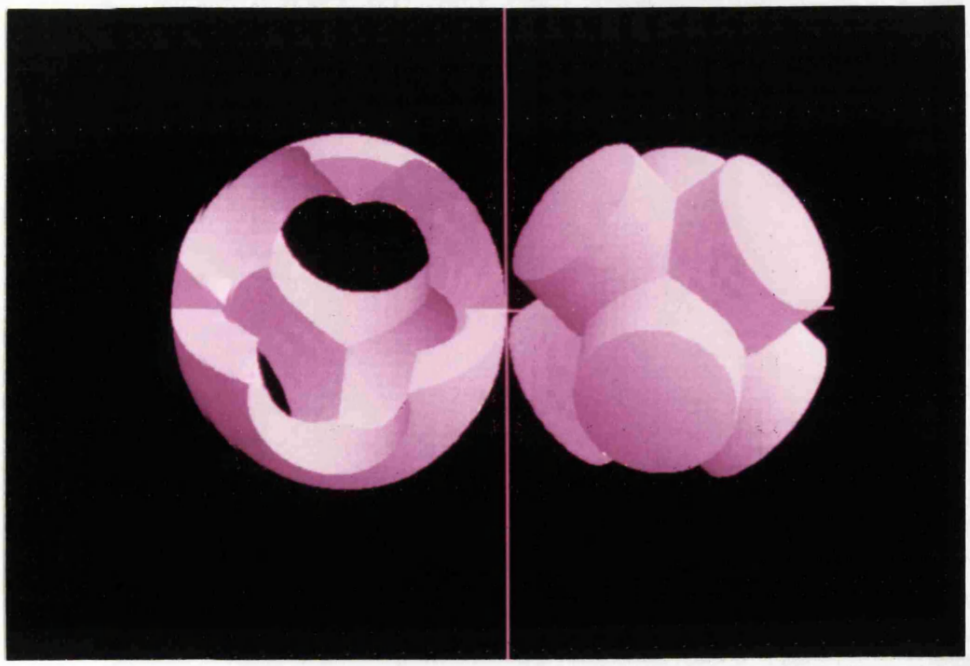


Fig.7.5 Polygon Models (2)

Wire-frames are also line drawings. They show the shape of the model by its outline and the edges between different surfaces. Though they are not as realistic as shaded pictures, they are geometrically more descriptive and give the user a better sense of the topological and spatial structure of the model. They also allow the user to view the inside or or reverse of the object without removing the front part (if hidden lines were not removed), which is very useful for interactive operation on the model, though they may sometimes be ambiguous [49]. Hidden edges can be removed to give better views of a 3D object [56] [66][90]

It is much easier to move a wire-frame than a shaded picture on the screen. This makes wire-frames suitable for robotic or NC machining control simulation or monitoring and other similar operations. It is relatively slow to generate a wire-frame view from a CSG model in comparison to a B-rep model, but with appropriate algorithms [109] such pictures can be generated fairly quickly for models made of simple geometric entities. It still remains slow for complicated surfaces. Fig.3.41 shows a wire-frame view of a CSG model made of curved surfaces. This pictures is generated as a by-product of the ray casting operation for determining the regions of different parts of the object in order to produce an efficient cutter path for NC machining [Chapter 8].

7.2.4 Realistic Shaded Pictures

Techniques for generating realistic pictures from geometric information about a model have been developed [19]. Among them, ray casting techniques are increasingly used to produce realistic pictures for complicated models. Ray casting is a very powerful technique for picture generation. Different ray casting techniques have been used to generate very sophisticated images [2][22][23][43] [75][91][93]. It can simulate a camera to produce an image that will look real. For some applications, if the geometry is the main concern and other factors are less important, a simplified picture with only colouring and shading effects can be produced, as opposed to including shadow casting, specular reflection, translucency, etc.

In the system described in this thesis, to keep the consistency between images and the model, and also to enable the same technique to be used in generating data for producing NC machining instructions, parallel ray casting was used to generate shaded pictures. Most of the shaded pictures for this thesis were generated in this way.

7.3 CALCULATIONS FOR RAY CASTING

The key problem for ray casting is the calculation of the intersection point of the ray and a surface. This, to a great extent, determines the speed of picture generation. For plannar surfaces, this calculation is straightforward, but for a curved surface it becomes difficult. Analytic methods for calculating intersections between low order surfaces have been proposed, but for higher order surfaces, numerical methods still have to be used.

In the system described here, models are represented by polynomials, so the problem becomes one of root finding for the polynomial.

Developing an efficient and reliable method of root finding for all kinds polynomials used in this system was beyond the scope of the project (which is still being researched at Bath University). For simplicity and generality, a numerical method, the *binary search* technique, was used in calculating intersections between the ray and a polynomial.

Binary search used only the potential value of polynomials, so the calculation was relatively simple and stable. The basic idea of binary search for a root is that, given two points in space, if their potential values have different signs, then there must be at least one root between them. By checking further the sign of the potential value between them, we can then decide which side the root lies and hence throw away the other half. By performing this procedure recursively until the distance between the remaining two points is small enough to meet the precision requirements, the middle point between them can be taken as the intersection of the ray and the polynomial.

One drawback of binary search is that the distance between the initial two points has to be small for curved surfaces, because this method can only guarantee to find one root between two given points. Since the existence of a root between two points was judged by the signs of their potential values, if a polynomial has a corner (the apex of a cone, for instance) and it happens to lie between these two points, the signs of the potential values of

the two points would be the same, and the binary search would fail to find a root between them. (This situation could be detected by also checking the direction of the gradients at the two points.)

To perform binary search, fist we must define the starting and ending points for the ray. Second, the ray between the starting and ending points must be chopped into small segments. The search was carried out on each of the segments from the starting point towards the ending point until an intersection was found, or until the ending point was reached.

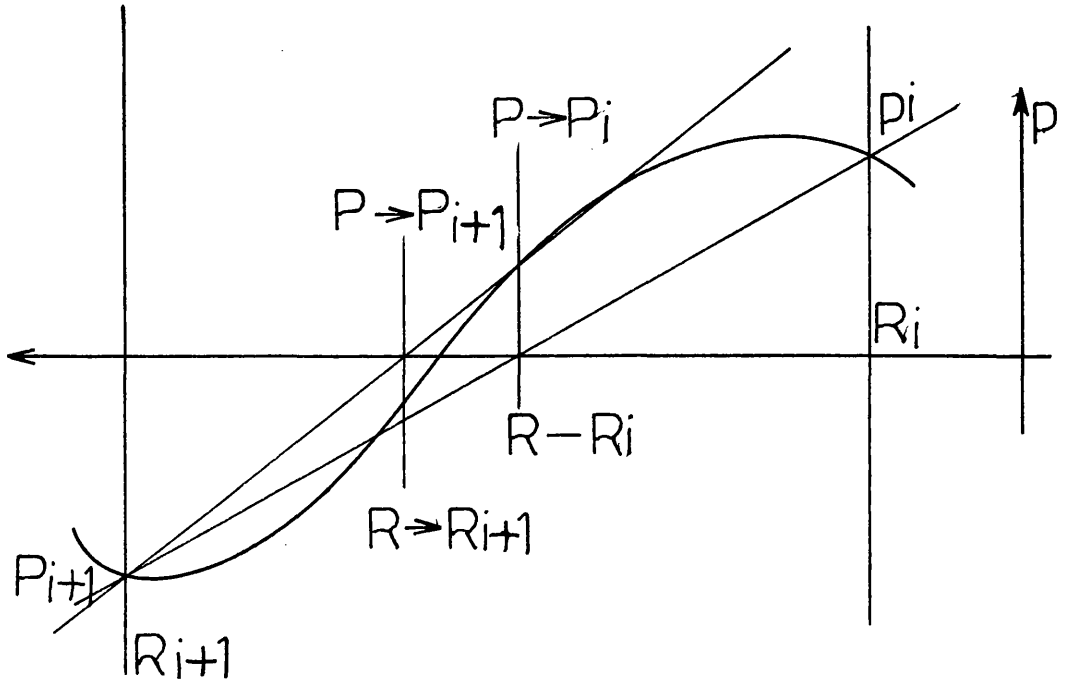
In the system described here, a modified binary search algorithm was used in calculating the intersection between the ray and a polynomial. This method used the fact that, for a polynomial, the absolute potential value of a point was greater than that of another point which is closer to the surface of the polynomial. Therefore, instead of chopping the segment in two parts of equal length, uneven chopping of the segment was used by making use of the potential values at the two ends of the segment (Fig.7.6). This made the process more efficient.

To illustrate the method, let us consider a single half-space, $F(R)$. Suppose that the calculation was carried out along the ray R from the starting point to the end point. All the successive segments along R were marked as $R_1, R_2, \cdots, R_i, R_{i+1}, \cdots, R_n$. Their corresponding potential values were $P_1, P_2, \cdots, P_i, P_{i\,1}, \cdots, P_n$. Following the steps described below, the first intersection between the ray and the half-space could be found.

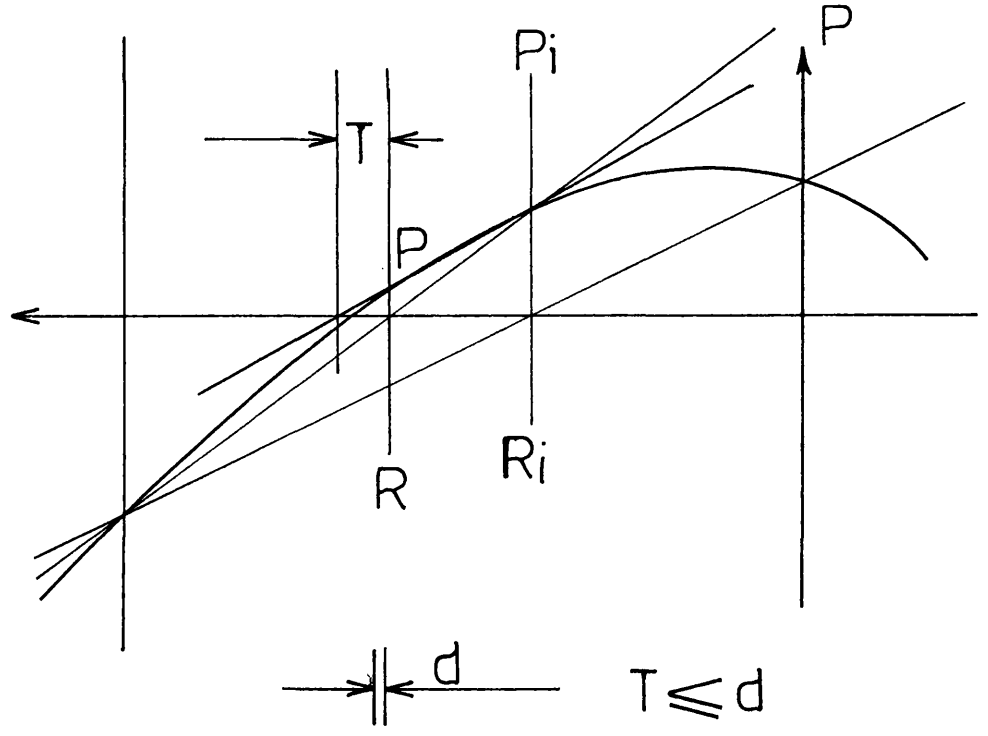
1. Locate an intersection by successively checking the potential of each increment along the ray. If the last two potentials, P_i, P_{i+1}, have different signs (P_i positive and P_{i+1} negative, for instance), an intersection is assumed between the last two points, R_i and R_{i+1}.

2. Locate the position r of the intersection according to the values of the last potentials calculated

$$r = R_i - (R_{i+1} - R_i)\frac{P_{i+1}}{P_i - P_{i+1}} \tag{7.3}$$



(a) Intersection Calculation



(b) Precision Checking

Fig.7.6 Ray Casting Calculation

3. Calculate the new potential P at r,

$$P = F(r) \tag{7.4}$$

4. If $P = 0$, r is the intersection.

If $P > 0$, and $P \geqslant P_i$ then $R_i = r$, $P_i = P$, goto **2**

If $P < 0$, and $P \leqslant P_{i+1}$ then $R_{i+1} = r$, $P_{i+1} = P$, goto **2**

5. Otherwise, calculate the precision distance T.

$$If\ P < 0,\ T = (R_{i+1} - r)\frac{P}{(P_{i+1} - P)} \tag{7.5a}$$

$$If\ P > 0,\ T = (r - R_i)\frac{P}{(P_i - P)} \tag{7.5b}$$

6. Checks the precision T against the user defined distance d.

If $T \leqslant d$, then r is used as the intersection.

Otherwise

If $P < 0$ then $R_{i+1} = r$, $P_{i+1} = P$, goto **2**

If $P > 0$ then $R_i = r$, $P_i = P$, goto **2**

The value of d can be different for different applications. It should be more precise in generating data for NC machining instructions than that for graphics displays.

The ray casting operation can be done in any direction. In relation to the Cartesian coordinate system, the position r effectively means (X, Y, Z) and the calculation

$$R_i - r$$

is effectively same as

$X_i - X$, $Y_i - Y$ *and* $Z_i - Z$

This method is much quicker than the even chopping binary search.

7.4 MEMBERSHIP TESTS

The intersections calculated using the above method are on the surface of each half-space concerned, but they are not guaranteed to be on the object's surface, since some sub-sets contain more than one half-space, particularly at corners where several half-spaces intersect each other. Before the ray reaches the object's surface, it may hit some half-space and hence generate an intersection not belonging to the object's surface.

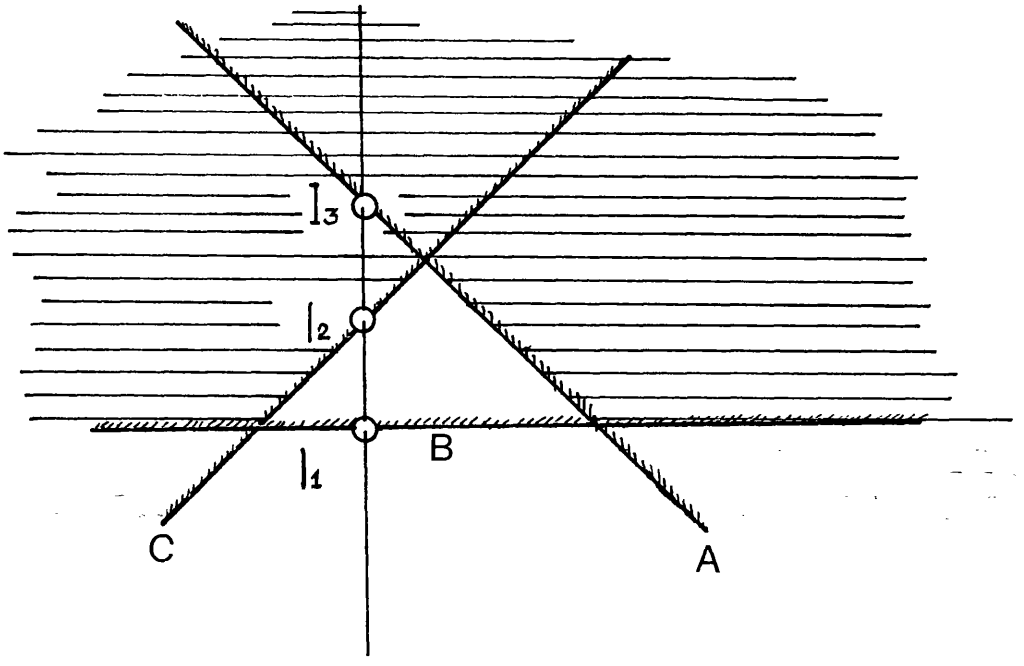
As an example, consider the sub-set shown in Fig 7.7. Three intersections are calculated along a segment of the ray. They belong to three different half-spaces. If these half-spaces are linked together by $B \bigcap (A \bigcup C)$, the second intersection will belong to the object's surface; if they are linked together by $A \bigcap B \bigcap C$, the first one will be on the object's surface. These half-spaces are exactly the same for both of the sub-trees, but they contribute to the object's surface differently if the set theoretic operations are different.

To decide which of the three intersections is the true one, they have to be checked against the sub-tree. This is called a *membership test*.

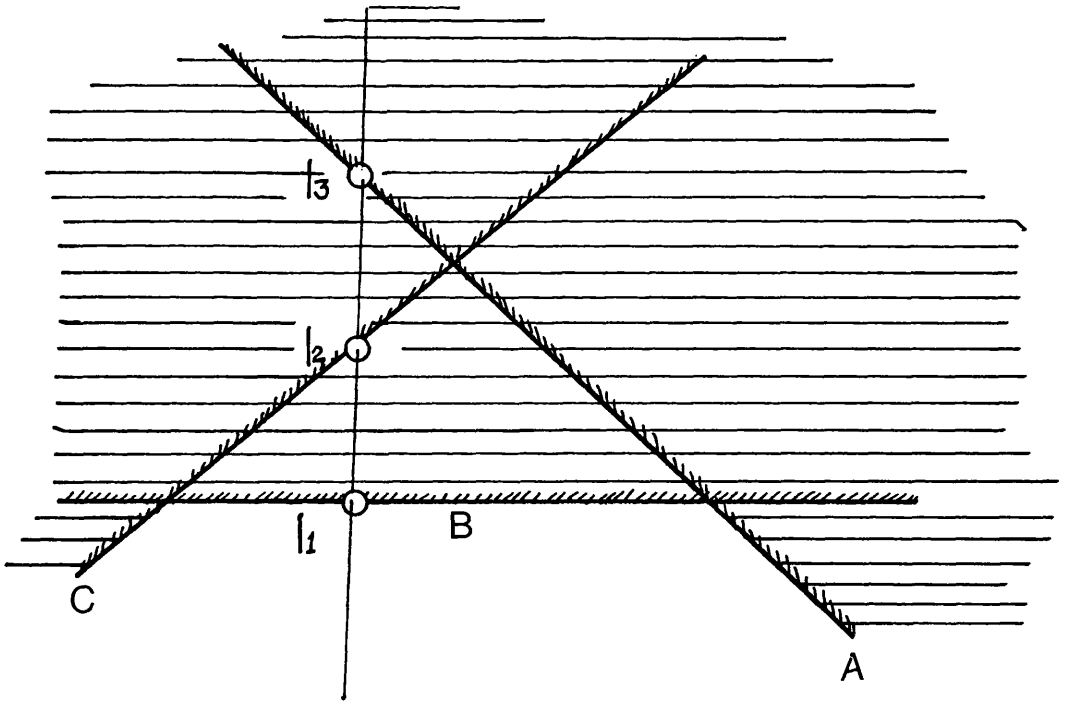
Let us take $B \bigcap (A \bigcup C)$ as the set-theoretic expression of the sub-tree. The potential values of all the half-spaces that belong to the same sub-tree needed to be calculated for each of the intersection points. The potential value for the half-space which was used in calculating the same intersection point was set to 0, others were calculated using the half-space equations. By replacing the half-spaces with their potential values at these intersections (Symbolically they are represented as *AIR* for positive, *SOLID* for negative and *ON* for 0), we can clearly see the result of the three intersections tested against the sub-tree:

$$I_1 : ON \bigcap (AIR \bigcup AIR) \rightarrow ON \bigcap AIR \rightarrow AIR$$

$$I_2 : SOLID \bigcap (ON \bigcup AIR) \rightarrow SOLID \bigcap ON \rightarrow ON$$



(a) (A∪C)∩B



(b) A∪B∪C

Fig.7.7 Membership Test

$$I_3 : SOLID \cap (SOLID \cup ON) \rightarrow SOLID \cup SOLID \rightarrow SOLID$$

Having done the test, the second intersection is taken as the true point belonging to the object surface.

Once an intersection has been proved to be on the object's surface, the surface normal is calculated and, using equation (7.2), a shade number is calculated and assigned to that point, which may represent one pixel on the screen. By casting rays equal to the number of pixels of the *view-port* on the screen, a realistic picture can be produced.

CHAPTER 8

NC MACHINING INSTRUCTION GENERATION

NC machining and programming have been used for a long time, but it has not been until now, with solid modelling systems, that the complete automatic generation of NC instruction for cutting components with complicated surfaces has become, in principle, practical.

In this chapter, techniques are discussed for generating NC machining instructions automatically to cut components modelled by the solid modelling system described in this thesis.

8.1 BRIEF REVIEW OF NC MACHINING

The complete automation of production has long been a desirable object for engineers in various fields. Research and experiments to this end have been carried on ever since the beginning of industry. NC machining was invented because of the two basic requirements for the automation of production and the ability to handle complicated geometries. The complexity of the shape of components in many cases determines the automation level that needs to be achieved.

The simplest geometric problem is probably two-dimensional location, such as that involved in punching and drilling. Complete automation can be relatively easily achieved in such processes. Two-dimensional cutting is slightly more difficult, and is often met in cutting different shapes from sheet metals, where the main problem is the optimal use of the material.

Another class of components are the so called two-and-half-dimensional ones [58] (such as the bottom of a plug casing). The machining of this class of components can be done by iterating a number of two-dimensional cuts to form the different levels of the component. For each level, no vertical movement of the cutter is involved; it only occurring when moving from one level to another.

The most complicated components are fully three-dimensional objects consisting of different kinds of complicated surfaces and their combinations. More sophisticated techniques are required for the NC production of such components at a high level of automation.

Because of the difficulties with handling complicated geometry using built in microcomputers in NC machines, complicated NC programs are written on larger computers. On such a computer, much longer, more sophisticated NC programs can be written. These programs can then be transferred by, for example, an RS232 link, to the NC machine to control the machining process. Since no paper tape is needed in the

transfer of the NC program from the computer to the machine, this is called Direct Numerically Controlled (DNC) machining.

One important advantage of DNC is that it can make use of geometric data generated by other systems, such as a draughting system, for example. This means that more complicated geometry can be handled. Working with the computer interactively, the user can check and modify the geometric data and specify the relevant tool paths to cut them.

Many NC programming packages have been written and some of them are widely available. APT, COMPACT II and GNC are some of the well known NC programming packages [32]. Most of them are linked or integrated with draughting systems or other CAD/CAM systems. Attempts have also been made to use the data generated by solid modelling systems to drive NC machines [115,116] [13]. These enable full three-dimensional geometry to be accurately described and defined. However, as far as NC programming is concerned, most of them require the user to interact with the computer to make certain decisions about the geometry, tool path and other operations.

The automation of NC instruction generation is rather hard work and still contributes only part of the whole production system in industry [42][28]. It needs big data bases, sophisticated software and relatively high computing speed. With a CAD/CAM system equipped with a solid modeller, this becomes possible. As was discussed earlier, a solid modelling system has the power to hold the entire description of one or more engineering components. This information, particularly sufficient and accurate geometric information, enables various design, analysis and processing planning activities to be carried out. Anything that might be needed to generate NC machining instructions can, in principle, be done with such a system.

Conceptually, there are two changes from conventional NC programming to NC programming with solid modellers and other CAD/CAM systems.

Firstly, an NC machine used to be considered as a more or less independent device, but when driven by a solid modelling system, it is better considered as a computer peripheral, just like a terminal or a plotter. The majority of the complicated tasks are done by the solid modelling system implemented on the more powerful computer, the NC machine's microcomputer controller being delegated to following a simple set of movement instructions.

Secondly, conventional NC code preparation used to take several steps to complete. These included the definition of the component's geometry, specification of the tool parameters, cutting speed calculation, cutter path programming, post processing and paper tape punching. This way of producing NC code is tedious, and less automatic and effective than it could be. However, with the system described above, all activities from design to the manufacture of the component are integrated in one system. This simplied the process a great deal and some steps (such as paper tape punching) are no longer necessary [40]. With such facilities and techniques, much more complicated tasks can be done more quickly, reliably, conveniently and automatically.

8.2 THE MACHINING ENVIRONMENT

The model of an object constructed by the solid modelling system described in this thesis is a geometrically complete three dimensional model. Theoretically, this information held on the model can be used to drive any kinds of NC machines with any number of axes.

In the development of this solid modelling system, a three-axis milling machine was available for the experiments. The three-axis mill, a Matchmaker, was controlled by a Fanuc Systems 6M - Model B controller [112], a high-accuracy, high-performance fixed-software CNC for milling machines. When machining, the controller could accept instructions from its memory, paper tapes or directly from a computer, as was used with the solid modelling system. The link between the computer and the NC controller was achieved using a BBC microcomputer, which transferred, checked and controlled the data flow from the main computer to the controller.

NC code generation used as its data source the octree described in Chapter 6 and generated G codes to drive the NC machine directly.

8.3 PROBLEMS IN NC CODE GENERATION

NC instructions can be used to control almost all general purpose machining processes, such as drilling, turning, milling, punching and nibbling, wire spark erosion, flame and plasma cutting and laser cutting. These are different in their nature and complexity. Among them, milling machines have been used for NC machining more widely, specially for cutting components with complicated surfaces. A milling machine usually has three or more axes. Different cutter shapes and sizes can be used on a milling machine. With proper fixtures different kinds of machining can be carried out. Generally speaking, a NC milling machine is more flexible and versatile than the others.

As a milling machine has these advantages in NC machining, (and because of the fact that a three-axis NC milling machine was actually available in the laboratory,) the discussion on solutions on various problems in NC machining will be mainly concentrated on three-axis milling machines.

There are many problems that need to be considered for the automatic generation of NC machining instructions from a solid model. In this thesis, since the aim was to prove the capability of the solid modelling system in handling complicated geometry and in automating the NC code generating process, the followings were considered in designing the system for generating NC instructions for the three-axis mill.

a. **Surface Complexity:** As was discussed earlier, geometric complexity affects the automation level. To achieve a high level of automation, the surface complexity restriction should be removed. In this thesis, the surface type was generalised to any half-space defined by implicit polynomials.

b. **Surface Combinatorial Complexity:** Most of the engineering components in mechanical engineering consist of two or more surfaces, which are defined independently by different mathematical equations. This combinatorial complexity has been one of the outstanding problems with many systems, where each piece of

surface had to be dealt with separately. In the system described here, the object's surfaces were considered to be a composite surface, and hence mathematically independent surfaces are not separated in NC code generation.

c. **Interference Avoidance:** This is one of the basic problems to be solved in NC programming. In the system described here, the avoidance of interference between the cutter and the object was handled automatically. This included the detection of undercuts, concave corners and internal surfaces, and algorithms to handle them.

d. **Tool Type Consideration:** To cut a complicated component, different parts need different cutters to achieve different surface finishing with high efficiency. The algorithm for NC code generation should be general enough to allow different types of cutters to be used.

The following section discusses some issues relevant to an attack on these problems.

8.4 THE AUTOMATIC GENERATION OF NC CODE TO DRIVE A THREE-AXIS MILL

The input to the presented system for the generation of NC machining instructions were the octree file containing the model, together with necessary machining parameters, such as cutter sizes and off-set for rough machining, which are specified by the user and contained in a command file which controls the running of the program.

Based on this input, the system generated NC machining instructions in the form of G codes, which were used directly to drive the three-axis milling machine to cut the desired surfaces.

The process of NC code generation divided into three main steps, which are discussed in what follows.

8.4.1 Two-Dimensional Grid Projection

The first step was to generate a two-dimensional grid from the three-dimensional model. The three-axis mill had only three axes with the cutter (the z axis) perpendicular to the x-y plane. The cutter could only cut surfaces that could be touched downwards without penetrating into or colliding with another part of the object's surface. Since any back surfaces (facing away from the cutter), under cuts (hidden by other part of the object's surface) and internal surfaces (forming the inside of the object) could not be cut, it was unnecessary to examine them to decide the actual tool path. Eliminating them from the model would reduce the amount of data and calculations, and would also reduce the chance for mistakes.

The two-dimensional grid included two groups of data: the basic grid and insert

points.

1. The Basic Grid

The basic two-dimensional grid was evenly spread in the x-y plane. It was generated by ray casting down into the three-dimensional model. The values at each node of the grid were the intersections of the ray and the upper surface of the model. Since the x-y grid was evenly arranged, their values were implicit, only the z values of the intersections needing to be stored. This reduced the amount of storage and resulted in a more compact data structure.

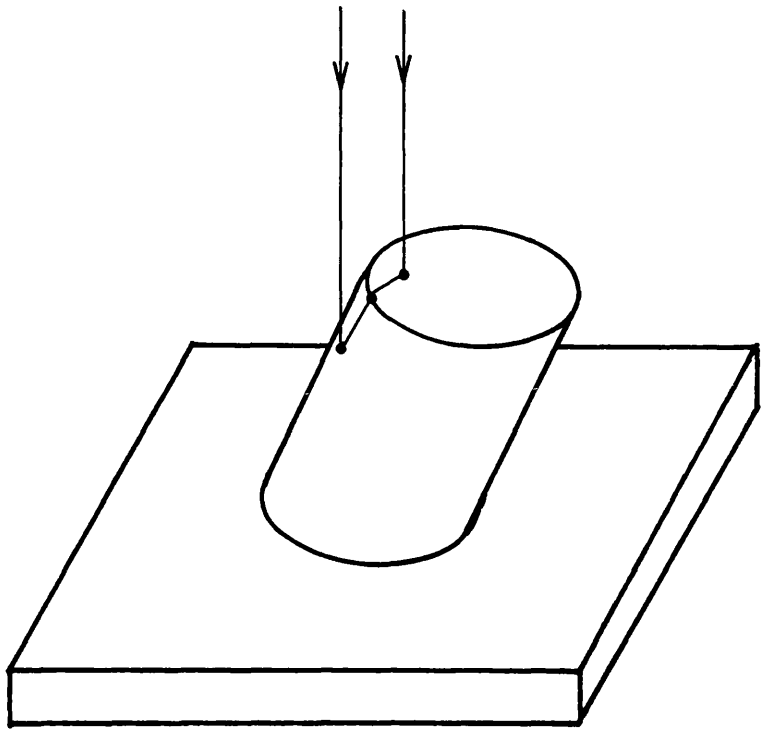
Because the octree sub-set model held the original definition of all the half-spaces and ray casting was used, the calculated grid nodes were exactly on the object's surface. In other words, at this stage, no approximation had been introduced. By controlling the horizontal resolution of the two-dimensional grid, the data could be as close to the real surface as needed.

2. The insert points

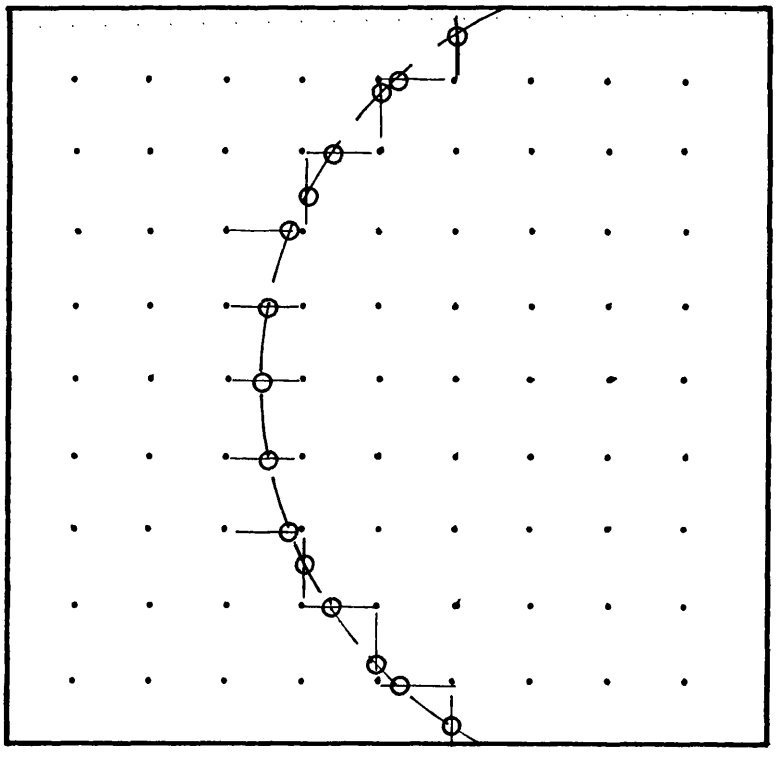
Though all the grid nodes were exactly on the object's surface, the horizontal distance between them would affect the precision of the representation by the two-dimensional grid. As an example, consider Fig.8.1, where the model consists of a cylinder and a plane. While the two-dimensional grid from ray casting could accurately represent the planes and the cylindrical surface, the edge and the outline of the cylinder could not be represented accurately. To represent the intersection curve accurately, x and y values need to be changed.

To improve the accuracy of the basic two-dimensional grid while keeping its simplicity, another group of data, called *edge points* or *inserted points*, were used to record points on the edges of each face of the object.

These edge points were calculated between grid nodes that were adjacent to each



(a)



(b)

Fig.8.1 Grid and Edge Points

other but did not belong to the same mathematically defined surfaces, such as the two grid points shown in Fig.8.1 (one on the plane and the other on the cylinder). The edge points were associated with their neighbouring grid nodes, but stored elsewhere.

The use of edge points improved the accuracy of the two-dimensional grid a great deal without reducing the distance between grid nodes. The basic grid and these edge points together formed the data source to be used for subsequent operations for NC code generation.

8.4.2 The Transformation to Cutter Height Grid

One of the basic tasks needed to generate a cutter path is to calculate the cutter position relative to the surface to be cut. There are several ways to decide on the cutter position, some of them are as follows:

1. Offsetting

This method decides the cutter position by offsetting from a surface point by the distance equal to the radius of the cutter in the direction of the surface's normal. Mathematically this is absolutely right, but in practice, the result may cause some problems.

a. Uneven spreading from curved surfaces causes cutter positions to be too far apart for convex regions, but too close for concave regions (Fig.8.2).

b. Because the normals of the object's surface do not always point in the same direction, evenly arranged surface points do not imply evenly arranged cutter positions in an x-y plane. As straight x-y line cuts are difficult to arrange for curved surfaces, this would cause more instructions for moving the cutter in x-y plane and would reduce the efficiency of cutting (Fig.8.3).

c. This off-setting may give difficulties in interference checking, specially for cutting composite surfaces, because the cutter position calculated in this way is difficult to relate accurately to other surfaces, and, if it is in a wrong position, such as in the

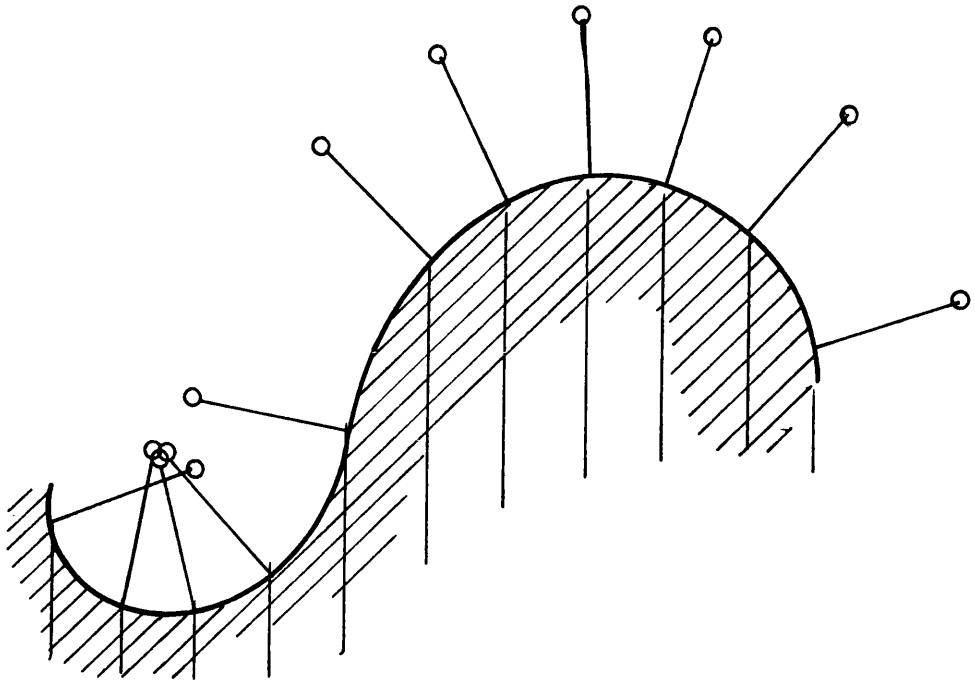


Fig.8.2

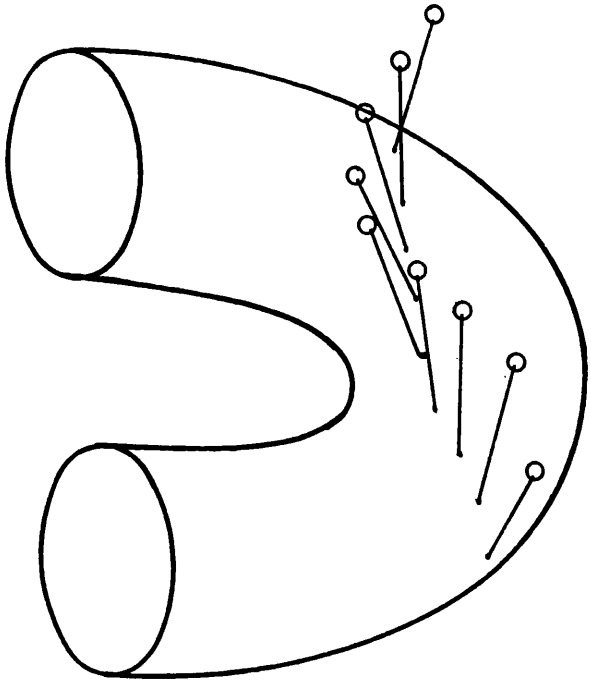


Fig.8.3

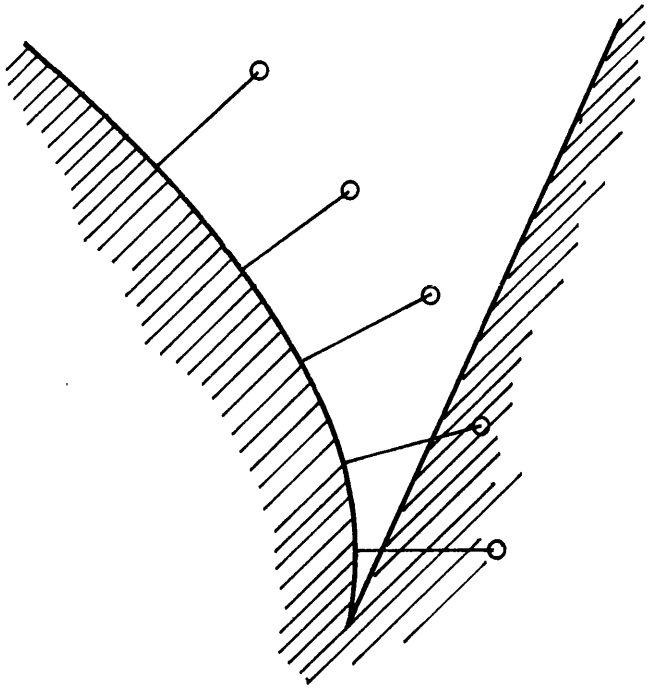


Fig.8.4

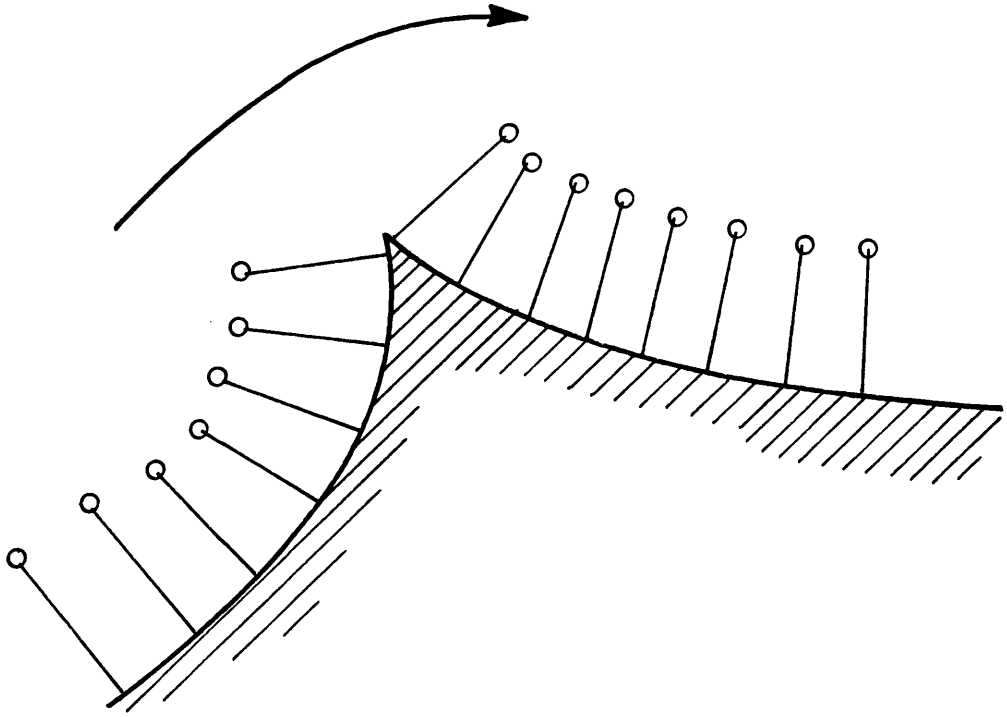


Fig.8.5

solid part of another half-space, it may be difficult to decide where it should be (Fig.8.4).

d. It may be difficult to deal with some sharp joints between surfaces where a continuous cutter path is required (Fig.8.5).

2. Driving Plane

Another way to decide cutter positions is to decide the cutter path in x and y first, then work out the cutter heights along this path. This is done by first calculating the intersection curves of the half-spaces and a plane, which is called the *drive plane*, then calculating the cutter positions within that plane above the intersection curve (Fig.8.6).

This method generates a cutter path with different z values following a straight line in x and y. However, apart from problems of interference detection, it also has its own problems.

a. The user has to define the driving planes for each straight line. This makes it difficult to automate NC code generation

b. This method can only handle one surface at a time. If a composite surface is to be cut, this method would difficult to use specially at blended areas, where surfaces are complicated and would make it more difficult to decide the starting and ending point for cutting each surface.

c. The cutter path is restricted to straight lines in the simplest form, though other drive surfaces could be used.

3. Grid Transformation

This is the method used in the system described in this thesis. This is a simple, different and effective method, and is used to handle the problems with the above approaches to cutter position calculation.

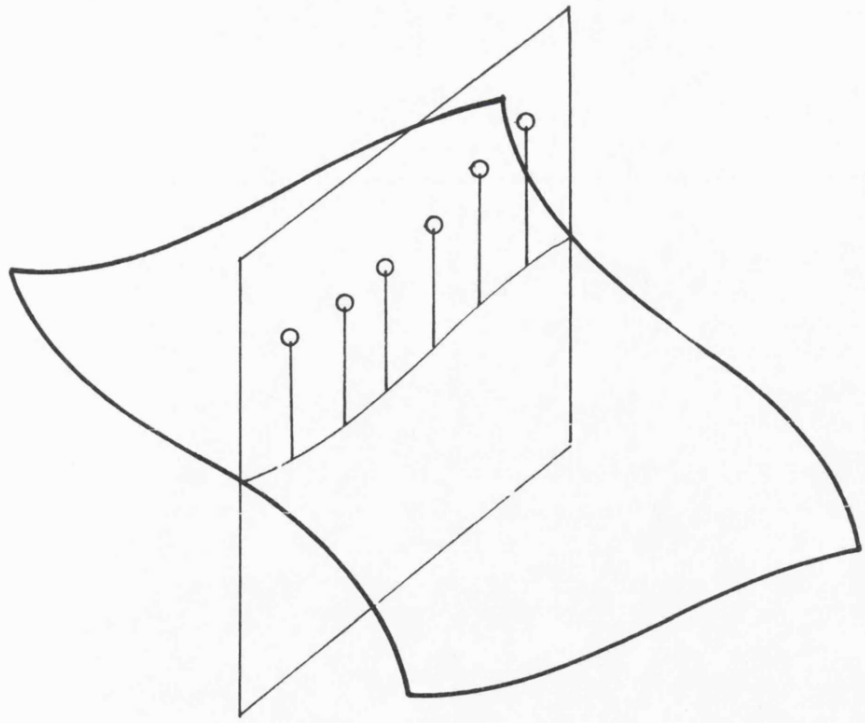


Fig.8.6 A Driving Plane

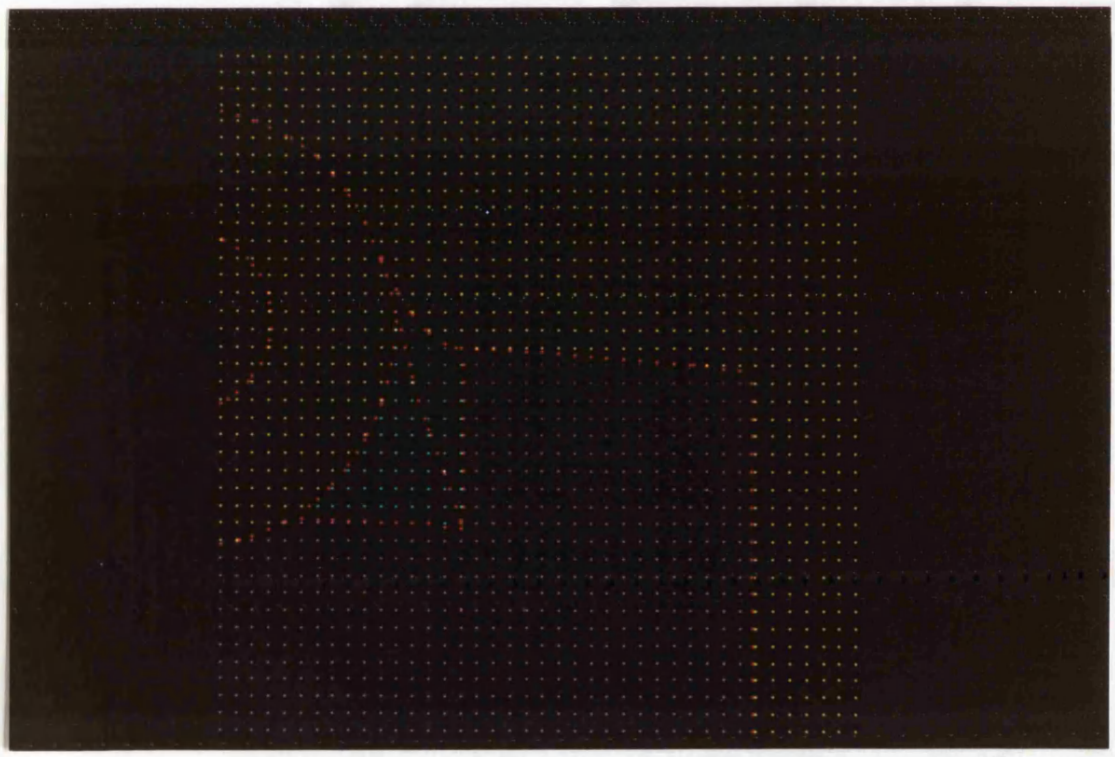


Fig.8.7 Part of the Grid of the Model in Fig.11

It was assumed that each of the grid nodes had a corresponding cutter position above it The cutter position had the same values of x and y, only z was changed. The z value for each cutter position is calculated using the following steps.

a. Choose the cutter shape. Several types of cutter could be used on the NC mill. Slot-drills and ball-nosed cutters were used for cutting objects shown in this thesis.
b. Decide the offset amount. This can be the difference between a roughing cut and a finishing cut or between several finishing cuts.
c. search all the grid nodes and edge points within the circular area determined by the cross section of the cutter, and decide a corresponding cutter height for each of them, the highest one of them all is the final cutter height at that node position (Fig.8.8).

The cutter height may (or may not) be decided by the corresponding node depending on the local environment, as shown in Fig 8.8.

The result of the transformation was another two-dimensional grid identical to the basic two-dimensional grid, except that the z values were changed.

The cutter position grid transformation method had the following characteristics

a. It generated cutter positions that were interference free between the cutter and the object.
b. It could accommodate different cutter shapes (ball-nosed cutters and slot drills were used in cutting objects for this thesis).
c. Definition of and controlling on offset were easy.
d. The generated cutter position grid could be arranged to form various different cutter paths.
e. Cutter positions were evenly spread in x-y plane for rough and finishing cutting, only vertical cutting and contouring followed curves of uneven x-y positions.
f. The object was considered as a whole and no user input was required.

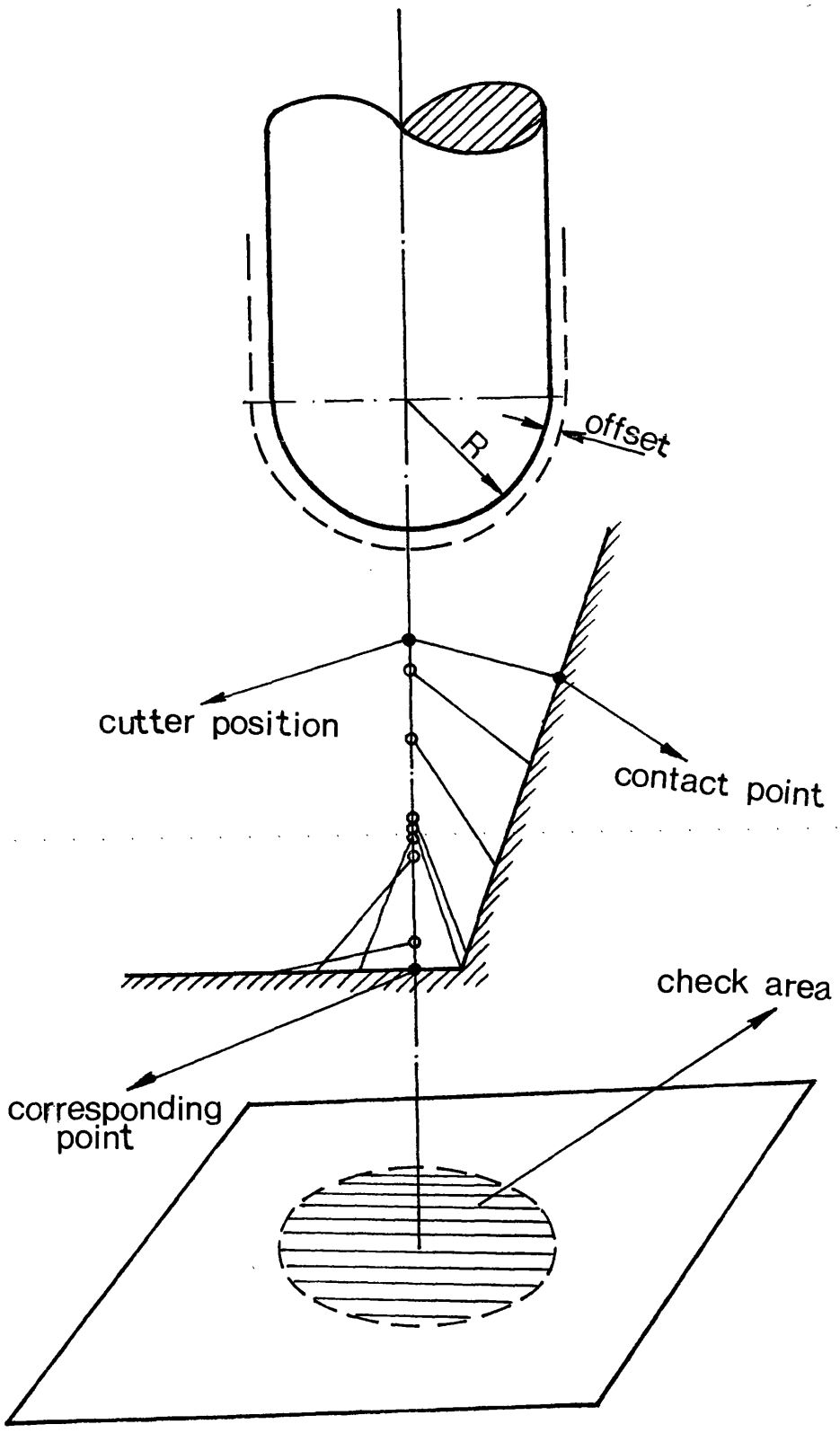


Fig.8.8 Cutter Height Calculation

8.4.3 Cutting Strategy Planning

Cutter path generating is not a easy job, even for simple geometries [68][84]. However, with the data structure used in this thesis, several kinds of cutting strategy could be derived from the cutter position grid and the edge points.

Based on information held on the cutter position grid, their associated edge points and surface type and regions, an efficient cutter path could be generated. The process of such generation could be entirely automatic, which demonstrated advantages over other systems.

To generate a cutter path that gave better quality of surface finish and cuts more efficiently, the surfaces that bound the object were classified according to their geometric characteristics and manufacturing requirements. The following were some surface types that were used in dividing the surfaces into different groups for different cuttings:

a. Horizontal Planes: planar half-spaces that were horizontal in the machine coordinates. They could have different heights. These planes were cut by slot drill to improve efficiency and surface finish.

b. Slope Planes: planes that were not horizontal in the machine coordinates. They were normally cut by ball-nosed cutters, but using simple instruction because one cut on such a surface needed only the positions of its two ends.

c. Curved Surfaces: this group included all kinds of curved surfaces and blends. They were cut by ball-nosed cutters in small increments to meet precision requirements and to achieved better surface finish.

d. Check surfaces: this group included all kinds of planar or curved surfaces, they distinguished themselves by their colour codes, which were assigned to them by the user during model construction. Their use was to help the system to calculated

materials to be removed in rough cut, and to stop the cutter from dropping too deeply into the object. Since they did not appear on the final object, no finishing cut was done to them. Fig.8.9 shows some check surfaces below a hand-wheel (Fig.8.10 shows the top view and the regional division).

Each group consisted of one or more different regions, which were bound by edge points. As show in Fig.8.7, the red points are edge points, which divide other grid points into different regions. The yellow coloured regions are planes and the others are curved surfaces and blends. Each small region represents a surface which is mathematically independently defined. Regions of different surface having the same manufacturing property were grouped together by their colour codes as show in Fig.8.7, so that they could be dealt with together. Check surfaces were marked using a colour band number by the user during the model's construction. Colour band numbers for other surfaces were automatically set up by the program according to their shapes and orientations. The following describes several ways to arrange the cutter position grid and edge points to form different cutter paths for different types of NC machining.

1. Rough Cut

As in many other kinds of machining, before the final fine cutting needed to produce the desired surface finish, relatively large amounts of material have to be removed from the raw material block.

The raw material block may have many different possible shapes. For simplicity, it was assumed to take the form of a cuboid, no matter what shape the final object might have. Roughing machining instructions generated under this assumption could be applied to any kind of object shape without affecting the final surface, though, for reasons of efficiency, modification and improvement were recommended. For example, the program may take two models as its input, one for the raw material and the other for the final components. The material to be removed can be calculated from the difference of these two models. From such information, more efficient rough cut paths

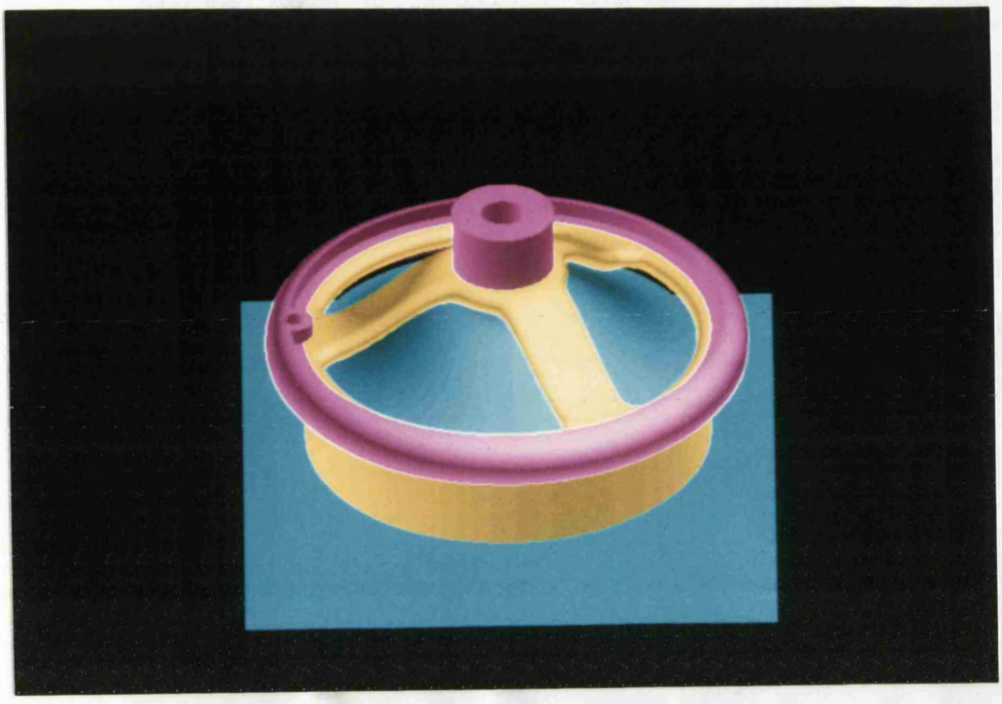


Fig.8.9 Cutting Position and Check Surfaces

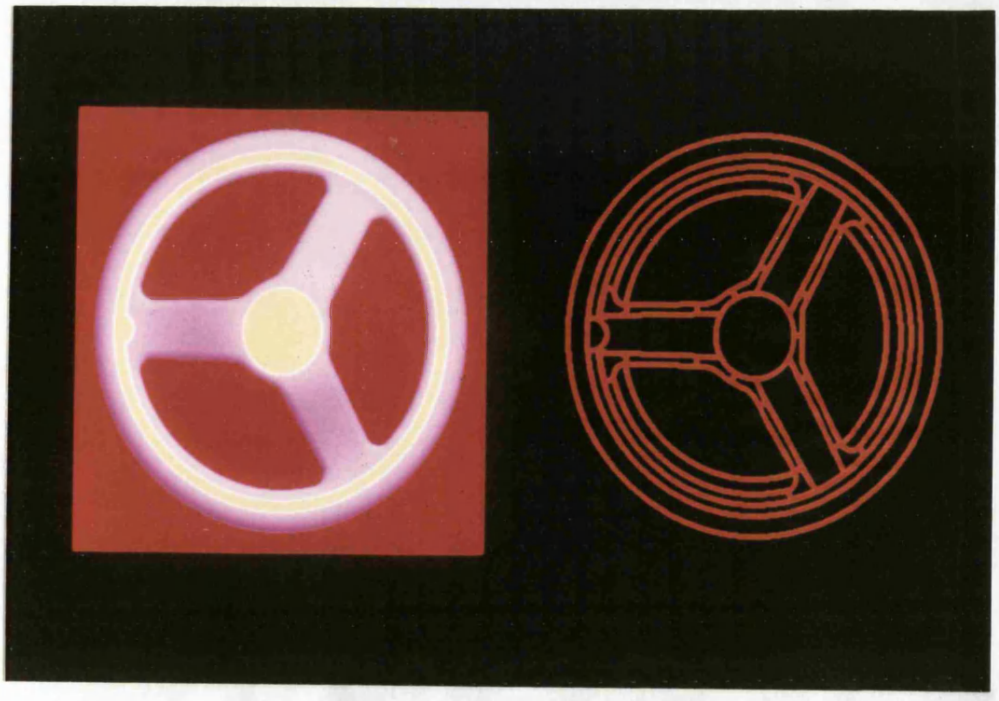


Fig.8.10 Top View and Regions

can be generated.

The basic job for roughing machining was to remove as much material as possible from the raw material block without damaging the final object's surface. Normally, a thin film of material was left for the finishing cut to remove. Another basic requirement for roughing machining was that it should be efficient.

To generate efficient roughing instructions, the cutter took the simplest route: straight lines along x or y. As the cutter position grid provides a free grid mesh for the cutter to move about, the cutter could just pass though all the grid nodes sequentially, line by line, either in x or y. The result would be close to the final surface.

However, as the distance between the grid nodes was very small, the material removed in one cut would be too thin for the roughing to be efficient; also, as the final surface might be very different from the raw material block the material to be removed in one cut would be too thick vertically to be cut.

To improve the behaviour of the roughing cutting, horizontally, the increment from one cut to another was set to be greater than that of the distance between grid nodes. This increment distance might be close to the radius of the cutter; vertically, the cutting was done in different levels, the depth[+] of each level was determined by the material, cutter size, machine power, horizontal increment distance and feed-rate.[+] Fig.8.13 shows the rough cutter path for the object show in Fig.11. This view was generated by simulating the cutter movement on the screen using the same G code data. (This was very useful in ensuring the correctness of the cutter movement before transferring the instructions to the NC machine. Views for such purpose can be generated in different planes. Fig.8.13 was drawn on the x-z plane so that it was possible to see the cutter height more clearly.)

The roughing NC machining instructions have been proven efficient, accurate and

[+] provided by the user in cutting objects for this thesis.

suitable for cutting composite surfaces, including blend surfaces.

2. Overall Finishing cutting

The simplest finishing cut was done in the same way as the roughing cutting, except that, as the shape left by roughing was close to the final surface and also a good surface finish was required, the cutter was driven to pass though all the cutter position grid points sequentially. The horizontal increment was the same as the distance between grid nodes and the cutting was not divided vertically. In cutting the hand-wheel (Fig.3.26, Fig.8.9, Fig.8.10), this method was used. Since this method treated all surfaces alike, it was not efficient. In cutting another component (Fig.8.11), the following improved methods were used, which were more efficient and accurate.

3. Regional Cut 1: Horizontal Planes

Regions of horizontal planes were cut with a slot drill to improve efficiency and surface quality. Fig.8.14 shows the cutter path for cutting the planes of the model shown in Fig.8.11 and in Fig.8.12, which is the top of the component in the position for machining, the red lines represent the edge points, which divide the component's surface in to different regions.

4. Regional Cut 2: Curved Surface and Slope Planes

This was the most difficult region to be cut, because they had the most complicated geometric shape. A ball-nosed cutter was used to cut such regions. Grid points representing curved surfaces were selected according to their colour codes, then cutter paths were generated to cut them. In cutting the object shown in Fig.8.11, the cutter followed a scanning sequence on this region. In comparison with the overall finishing cutting described above, since this regional cutting ignored the other simple and non-significant surfaces (check surfaces, for instance), it was much more efficient. Fig.8.15 shows the cutter paths for cutting part of the curved surfaces of the component. The pictures were generated by a programme which took the NC machining

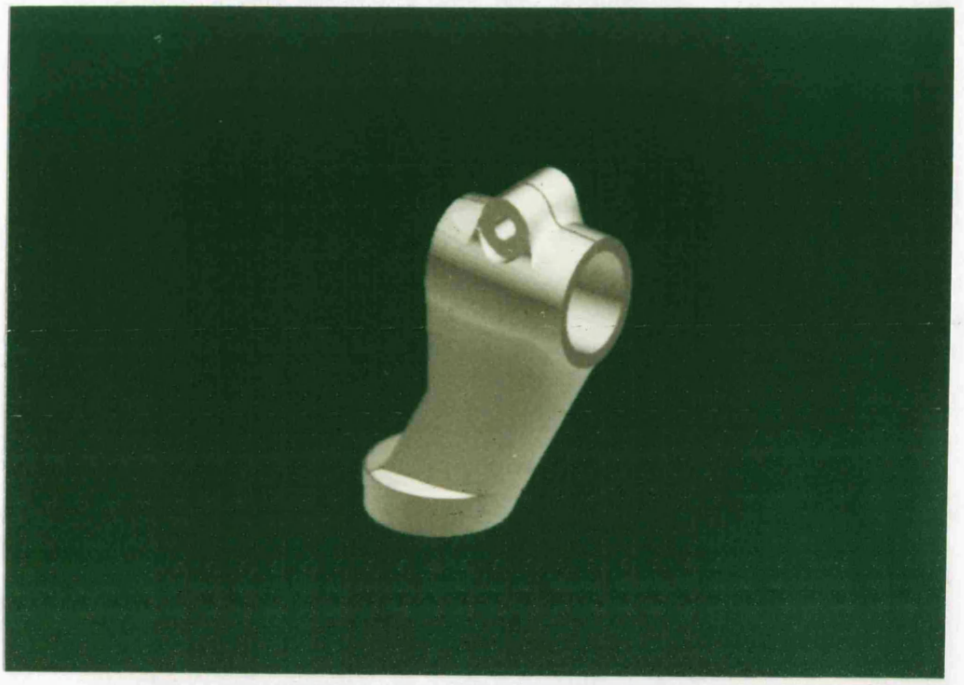


Fig.8.11 A Model of a Component

Fig.8.12 Top View and Regions for NC Machining

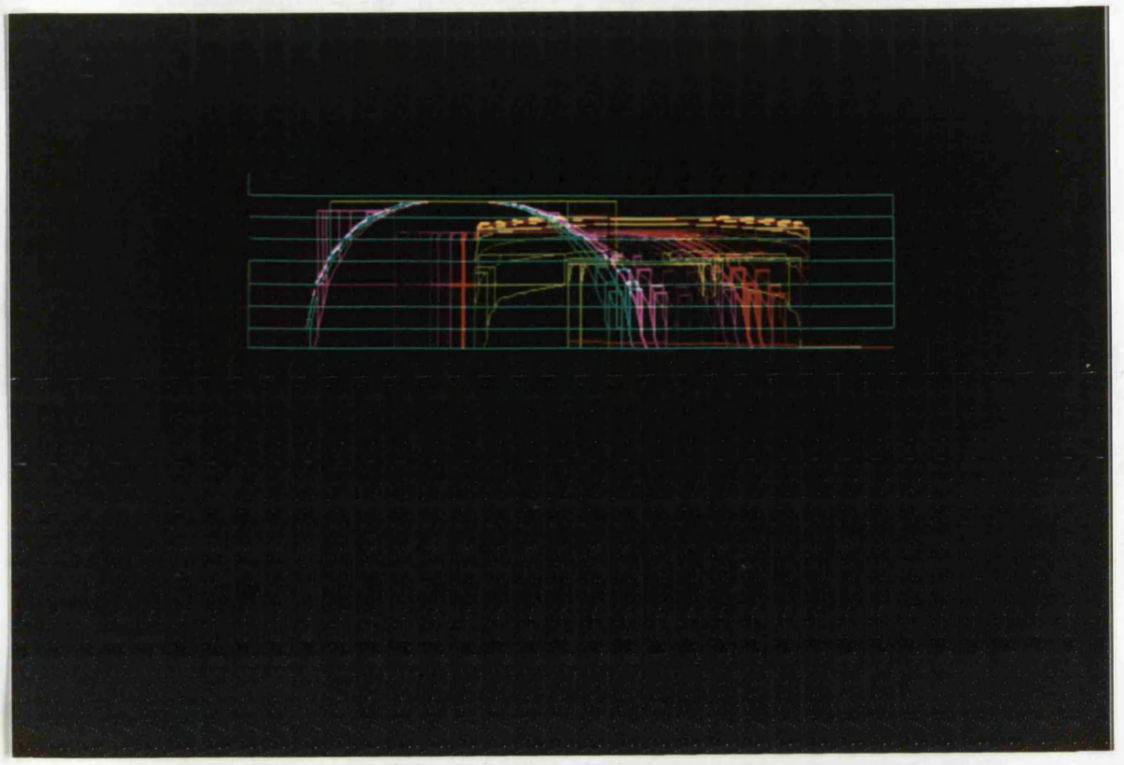


Fig.8.13 Cutter Path for Rough Cut

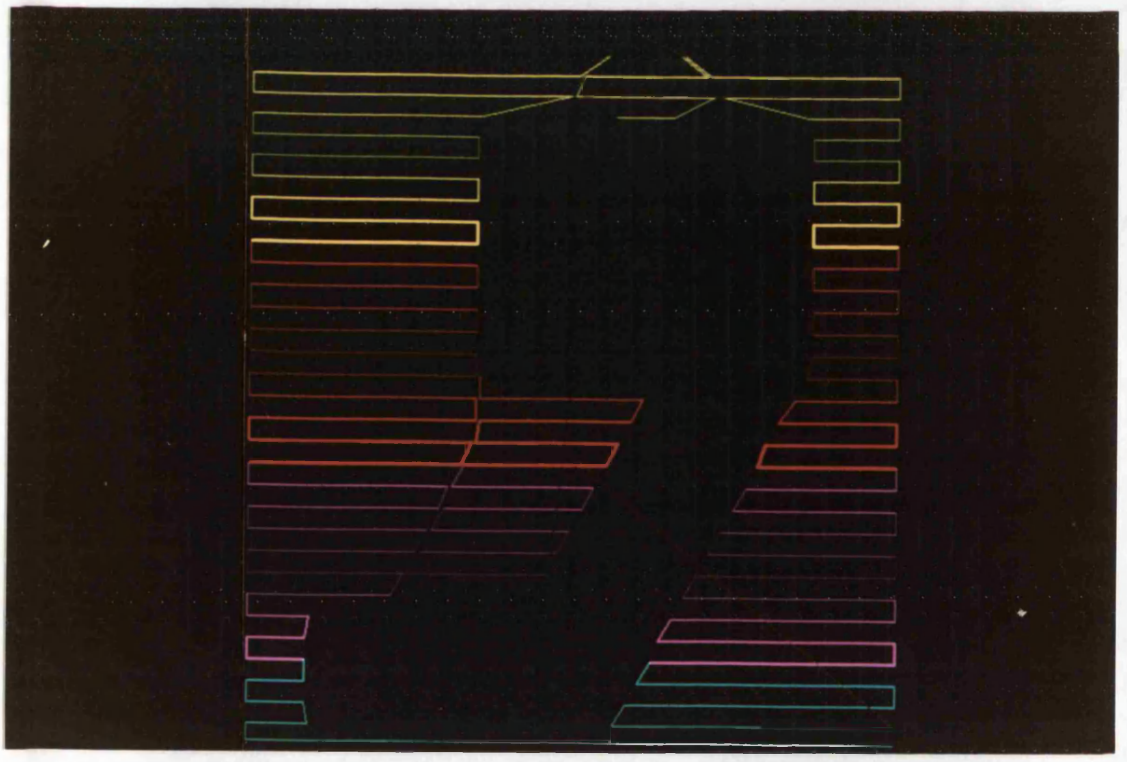
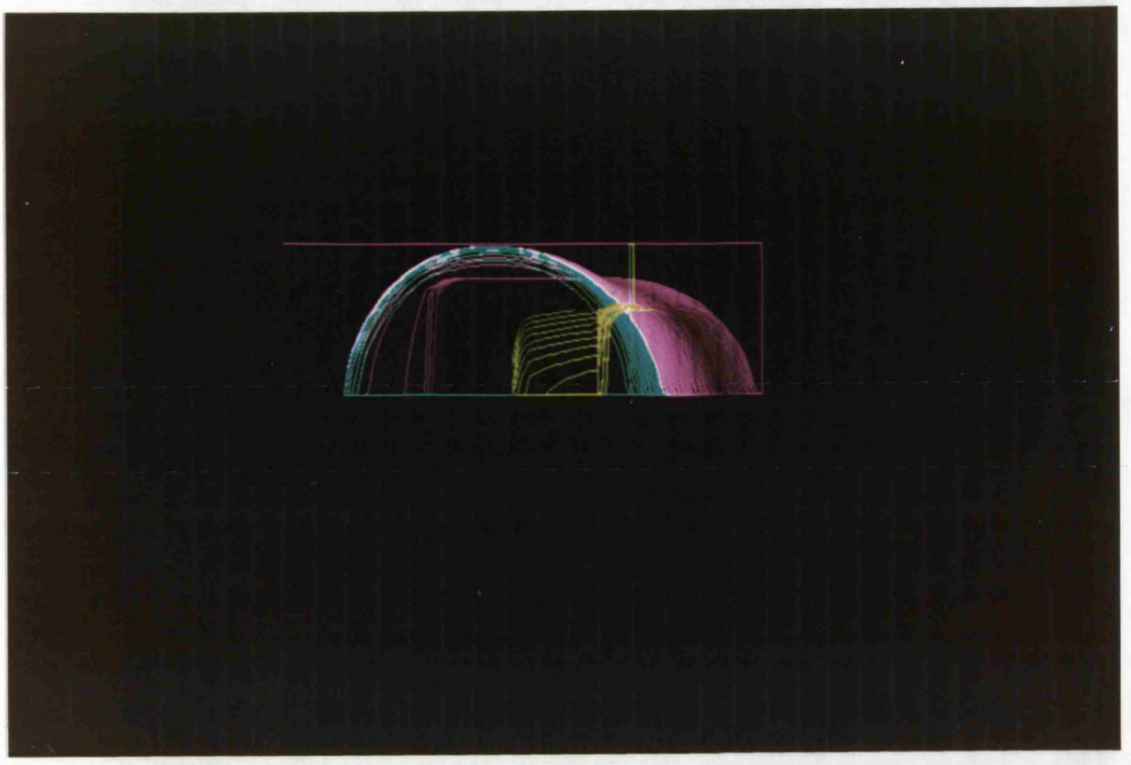
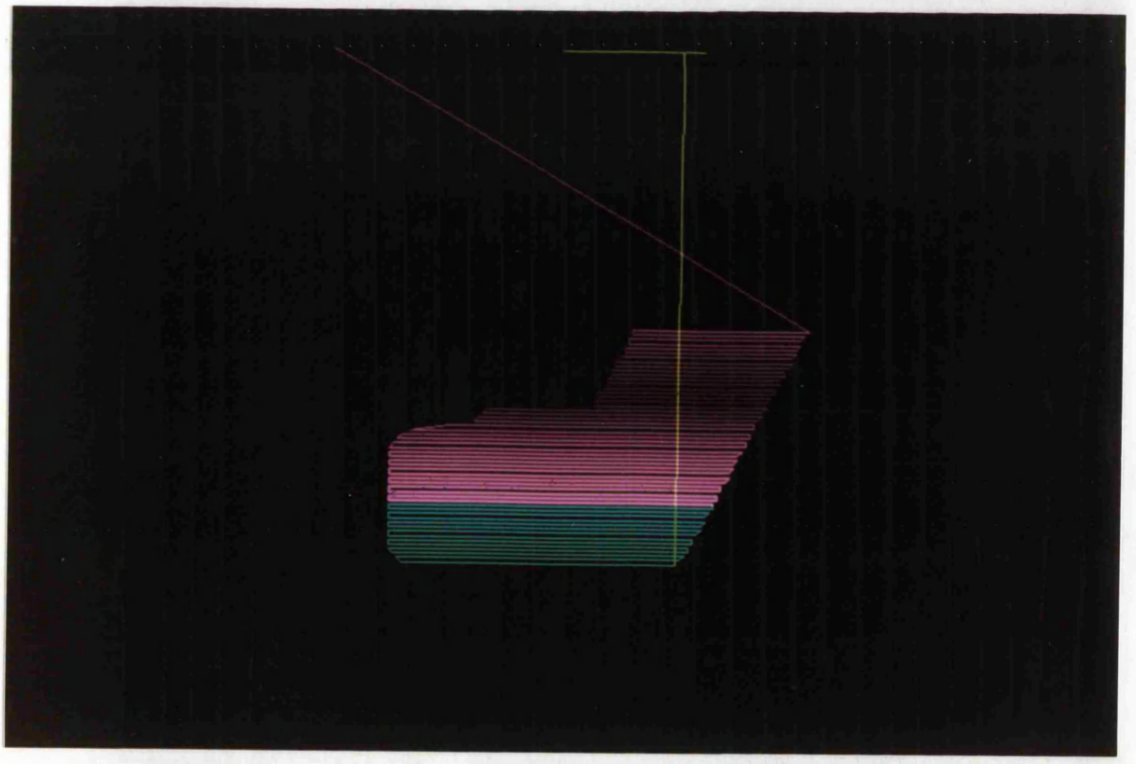


Fig.8.14 Cutter Path for Cutting Horizontal Planes



(a)



(b)

Fig.8.15 Part of the Cutter Path for Regional Cutting
Curved Surface of the Model in Fig.8.11

instructions and simulated the movement of the cutter. As shown in Fig.8.15, different views could be generated, which were very useful for checking the NC machining instructions before transferring them to the NC mill. Cutter paths other than in scan line sequence could also be arranged without difficulty.

5. Contouring 1: Horizontal Planes The regional cutting on horizontal planes described earlier removed most of the material, but some material was still left along the outline of the region. This left material was removed by contouring cutting along the edge points of the region, which divide the horizontal planes from other regions. Slot drills were used in this contouring cutting. The x and y values of cutter positions for the contouring cutting were calculated along the edge points, while the z heights were decided in the same way as for regional cutting. This gave accurate cutter position both in x-y and in z.

6. Contouring 2: Vertical Surfaces and Outlines

Vertical cutting was important to produce better vertical surface finish and to produce the outline of the object. Vertical cutting is arranged along edge points (in fact, edge points were the only data for locating vertical surfaces, because, the parallel ray casting could not produce intersections with vertical surfaces). The height of the cutter position can be decided in the same way as for the cutter position grid.

Check surfaces were used to stop the cutter from dropping down too deeply. They were very useful when a component was to be cut on both sides (like the hand-wheel in Fig.8.9). since check surfaces were not part of the final object's surface. They were considered the same as other surfaces in roughing machining, but were ignored in finishing cutting.

Either ball-nosed cutters or slot drills could be used for this type of cutting, depending what type surfaces were below the vertical surface or what type the check surfaces were. If they were horizontal planes, a slot drill would give the best result and

efficiency, otherwise, a ball-nosed cutter was used.

Fig.8.16 shows the hand-wheel being cut on the three-axis mill. Fig.8.17 is a picture taken after the rough cutting of the second side of the hand-wheel and Fig.8.18 shows the final shape of the hand-wheel. Fig.8.19 shows a machined object of the model in Fig.8.11. The hand-wheel was cut using the simple x-y scan line cutting, and no edge points were used in calculating the NC code, while the object shown in Fig.8.19 was cut using NC machining instructions generated using regional cutting algorithms, and edge points were used in deciding cutter positions and cutter paths for contouring. The second one was cut more efficiently, and the result was more accurate as can be seen from the pictures.

Fig.8.16

Fig.8.17 A Hand-Wheel After Rough Cutting

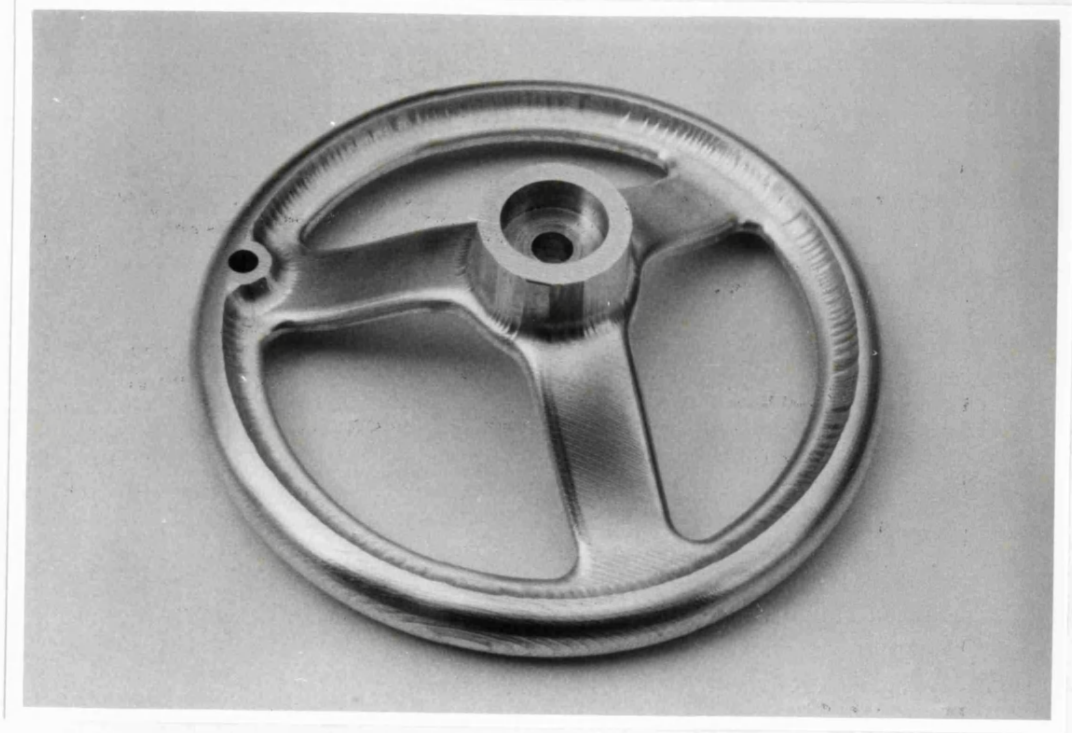


Fig.8.18 The Machined Hand-Wheel

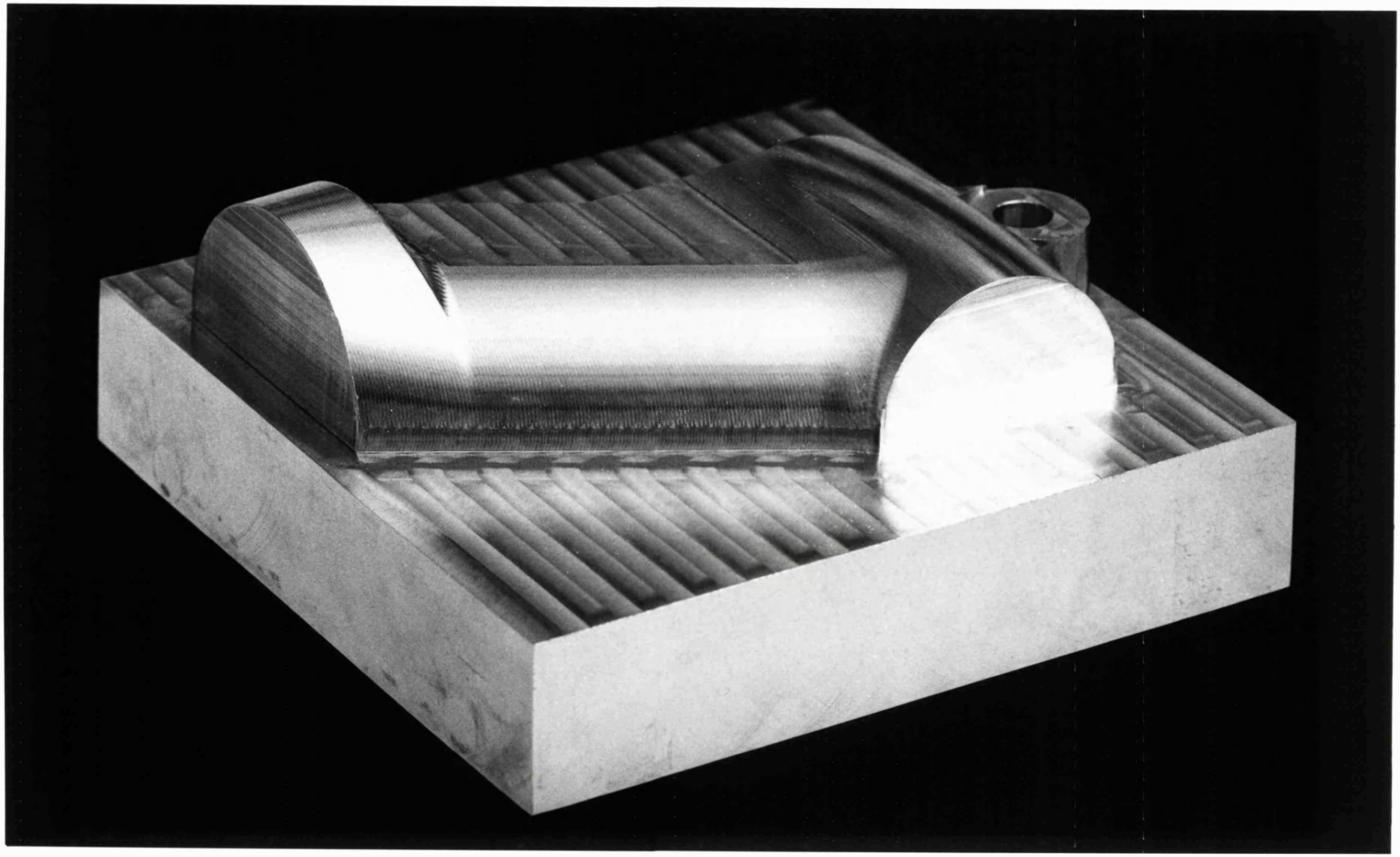


Fig.8.19

CHAPTER 9

CONCLUSIONS AND REMAINING PROBLEMS

9.1 AIMS AND ACHIEVEMENTS

Blends and fillets are important but non-critical regions of smooth surfaces between major features of a wide range of engineering components, but unfortunately they were missing from their models generated by many solid modelling systems, because of the absence of an appropriate blend representation.

Faced by the difficulty of integrating parametric *free form surfaces* with *simple primitives*, this research project was set up to investigate the suitability of various implicit polynomials in representing the shape of all kinds of engineering components, and their consistency and reliability in various engineering applications. The following are the main achievements of this research work.

a The multistage method of constructing implicit polynomials of different order from constants and planar half-spaces has proven to be very effective. It provides a tool to integrate various simple and complicated surfaces consistently in one system.

b The formulations for corner blend and gap blend were convenient to use and easy to control. The most important feature of them was the consistency between the blend and the blended surfaces, which makes them particularly suitable for representing engineering components.

c The applicability of the solid modelling system has been tested by producing real object from the the model of the object. Accurate and flexible NC machining instructions could be written automatically according to the geometric information

about the model and necessary manufacturing information given by the user.

Some other issues, such as representation schemes, data structures, and so on, have also been investigated. Fig.9.1 shows the main structure of the system described in this thesis. Research and experiments have shown that techniques used in this thesis were effective methods for representing and handling information about complicated engineering components. They are new and important contributions to CAD/CAM/CAE systems.

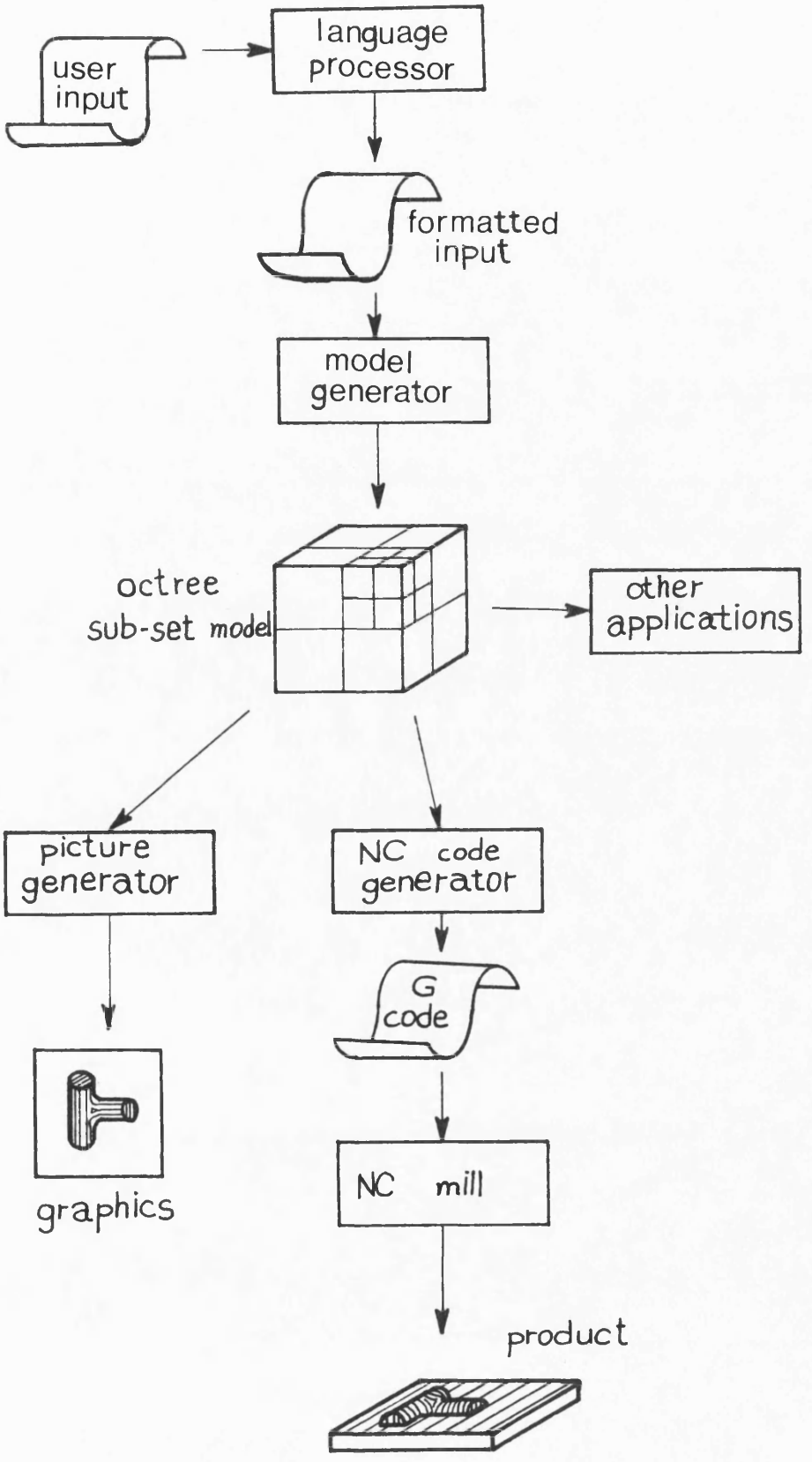

user input
language processor
formatted input
model generator
octree sub-set model
other applications
picture generator
NC code generator
G code
graphics
NC mill
product

Fig. 9.1

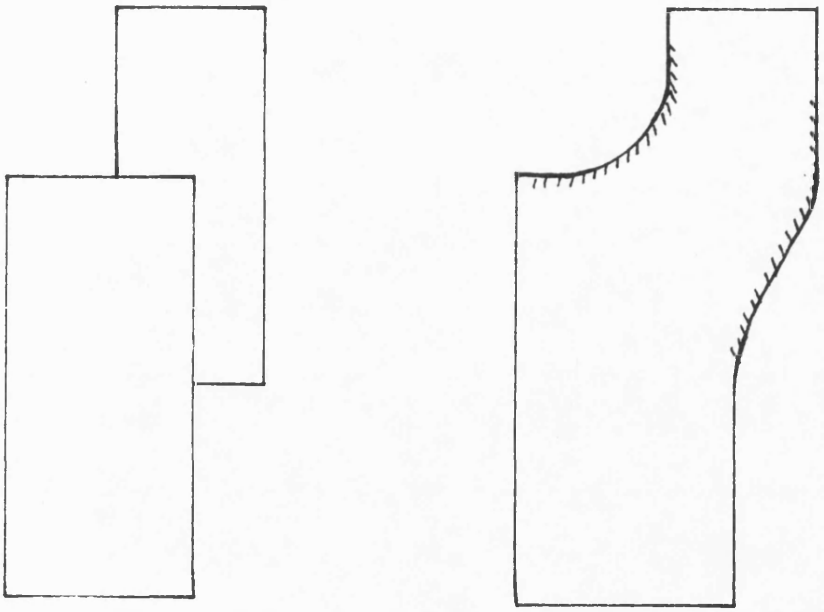
9.2 SUGGESTIONS FOR FURTHER WORK

Despite of the above achievements, this solid modelling system is not yet a mature one and it needs to be improved in several aspects to make it more accurate and efficient. The following are some remaining problems that have not yet been solved or implemented.

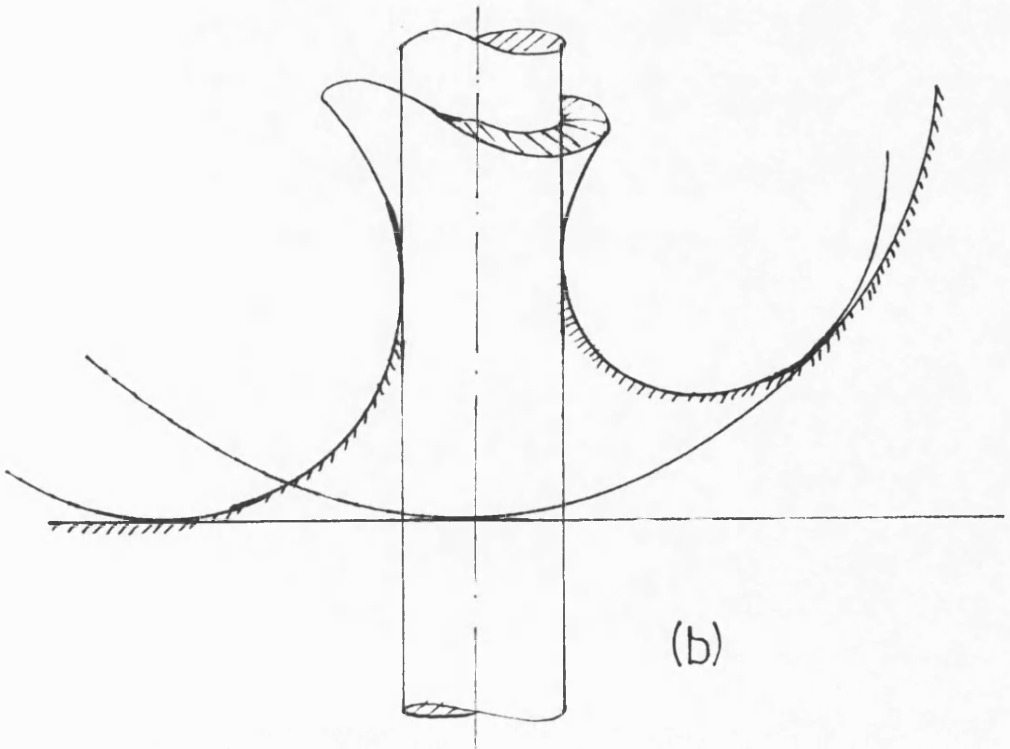
a In determining the relationship between a half-space and a sub-space, a more precise method is needed to replace the minimum maximum test, which is in fact an approximation approach and is far from accurate for high order half-spaces.

b Blend surfaces were generated on mathematically defined *single* surfaces. When a blend is required at junctions between two or more half-spaces, specially at the convex side as shown in Fig.9.2, the method used in this thesis may be difficult to employ.

c In NC machining instruction generation, effort was made mainly to attack geometric and algorithmic problems. For more sophisticated NC machining instructions, factors such as feedrate calculation and tolerancing should be included.



(a)



(b)

fig.9.2 blends at joints of different surfaces

APPENDIX 1

A FORMATTED INPUT FILE

The following is an example of a formatted input generated by the input language processor BLINP. The input file is the description of a model consisting of a cone, two spheres and blend surfaces between them (Fig.3.25), which was given as the example of input language BLINP earlier in Chapter 4.

Example of a Formatted Input File

```
1.0000,0.0000,0.0000,0.0000,2,   HX
0.0000,1.0000,0.0000,0.0000,2,   HY
0.0000,0.0000,1.0000,0.0000,2,   HZ
0.0000,1.0000,0.0000,-60.0000,6, HY101
0.0000,-1.0000,0.0000,0.0000,2,  HY102
-1.0000,0.0000,0.0000,-35.0000,6, HX201
1.0000,0.0000,0.0000,-35.0000,6, HX202
0.0000,1.0000,0.0000,-20.0000,6, HY201
1.0000,0.0000,0.0000,0.0000,8,   HX401
-1.0000,0.0000,0.0000,0.0000,8,  HX402
1.0000,0.0000,0.0000,-35.0000,8, HX403
-1.0000,0.0000,0.0000,-35.0000,8 HX404

1.0000000  1
2.0000000  2
0.5000000  SQR
0.3000000  CNS101
16.0000000 RAD201
10.0000000 RAD301
0.9000000  CNS401

*CON101
1,10002,-8,3,10002,-8,-4,4,10004,-6,10002,-8,-5;

*MDL1
4,5,-3,20001,-3;

*SPH201
6,10002,-8,8,10002,-8,-4,3,10002,-8,-4,10005,10002,-8,-5;
```

```
*SPH202
7,10002,-8,8,10002,-8,-4,3,10002,-8,-4,10005,10002,-8,-5;

*CYL301
8,10002,-8,3,10002,-8,-4,10006,10002,-8,-5;

*BLD401
10001,10007,-5,20001,-6,20003,-6,10007,20005,10002,-8,-6,-5;

*BLD402
10001,10007,-5,20001,-6,20004,-6,10007,20005,10002,-8,-6,-5;

*MDL401
20006,20005,-3,9,-3,12,-3;

*MDL402
20007,20005,-3,10,-3,12,-3;

*MDL4
20008,20009,-1;

*MDL5
20002,20003,-1,20004,-1,20010,-1;
0;
```

The first part of this example are the planar half-spaces, followed by the values of each constant, and then the definition of all the polynomials defined using the above planes and constants in the form of a reversed Polish expression. Set-theoretic expressions are used in the same way as arithmetic operators in the model description. Each string following a model name contains the arithmetic and set-theoretic definition of that model. Previously defined models can be used in subsequent definitions of other models. Information about graphics can also written in this file after the model's definition.

In comparison with the user description of the same model (Example 1 of input language), this is more compactly structured and hence is more suitable for the solid modeller.

REFERENCES

1 **Akkermans, M., Thillo, M. V., Gool, L. V. & Oosterlinck A.** *An Algorithm for the Extraction of Line drawings for Polyhedral Scenes and Their Use in Stereo Vision* Computers in Industry, North-Holland (1985)

2 **Amanatides, J.** *Ray Tracing with Cones* Computer Graphics, Vol.18, No.3, July (1984)

3 **Badler, N. I. & Fini, T. W.** *Computer Graphics and Expert Systems* IEEE CG&A, November (1985)

4 **Baer, A., Eastman, C. & Henrion, M.** *Geometric Modelling: a survey* Computer-Aided Design, Vol.11, No.5, September (1979)

5 **Barnhill, R. E.** *A Survey of the Representation and Design of Surfaces* IEEE CG&A, October (1983)

6 **Barr, A. H.** *Global and Local Deformation for Solid Primitives* Computer Graphics, Vol.18, No.3, July (1984)

7 **Barsky, B. A.** *Parametric Spline Curves and Surfaces* IEEE CG&A, February (1986)

8 **Beeker, E.** *Smoothing of shapes designed with free-form surfaces* Computer-Aided Design, Vol.18, No.4, May (1986)

9 **Besel, P. J. & Jain, R. C.** *Three-Dimensional Object Recognition* Computing Surveys, Vol.17, No.1, March (1983)

10 **Bez, H. E.** *Homogeneous coordinates for computer graphics* Computer-Aided Design, Vol.15, No.6, November (1983)

11 **Bin, H.** *Inputting constructive solid geometry representations directly from 2D orthographic engineering drawings* Computer-Aided Design, Vol.18, No.3, April (1986)

12 **Bloor, M. S., de Pennington, A. & Woodwark J. R.** *RISP - Bridging the Gap Between Conventional Surface Elements.*

13 **Bobrow, J. E.** *NC machine tool path generation from CSG part representations* Computer-Aided Design, Vol.17, No.2, March (1985)

14 **Boehm, W.** *Curvature continuous curves and surfaces* Computer-Aided Design, Vol.8, No.2, March (1986)

15 **Bowyer, A.** *SID: a language for describing solid objects* Software Manual, Bath University (1985)

16 **Bowyer, A. & Woodwark, J. R.** *A programmer's geometry* Butterworths (1983)

17 **Braid, I. C.** *The Synthesis of Solids Bounded by Many Faces* Communications of the ACM, April (1975)

18 **Brown, A. D. & Thomas, P. R.** *Improving the efficiency of scanline algorithms* Computer-Aided Design, Vol.18, No.2, March (1986)

19 **Carpenter, L.** *The A-buffer, an Antialiased Hidden Surface Method* Computer Graphics, Vol.14, No.6, November (1982)

20 **Casle, M. S. & Stanton, E. L.** *An Overview of Analytic Solid Modeling* IEEE CG&A, February (1985)

21 **Clark, J. H.** *Hierarchical Geometric Models for Visible Surface Algorithms* Communication of the ACM, Vol.19, No.10, October (1976)

22 **Cook, R. L.** *Shade Trees* Computer Graphics, Vol.18, No.3, July (1984)

23 **Cook, R. L., Porter, T. & Carpenter, L.** *Distributed Ray Tracing* Computer Graphics, Vol.18, No.3, July (1984)

24 **Crocker, G. A.** *Invisibility Coherence for Faster Scan-Line Hidden Surface Algorithms* Computer Graphics, Vol.18, No.3, July (1984)

25 **Crow, F. C.** *Summend-Area tables for Texture Mapping* Computer Graphics, Vol.18, No.3, July (1984)

26 **Crow, F. C.** *Shaded Computer Graphics in the Entertainment Industry* Computer Graphics, Second Edition, IEEE (1982)

27 **Deken, J.** *Computer Images State of the Art* First published in Great Britain in 1983 by Thames and Hudson Ltd, London.

28 **Dube, R. P. & Smith, M. R.** *Managing Geometric Information with a Database Management Systems* IEEE CG&A, October (1983)

29 **Duncan, R.** *Three-space hidden surface removal using boundary traversal logic* Computer-Aided Design, Vol.15, No.4, July (1983)

30 **Farouk, R. T. & Hinds, J. K.** *A Hierarchy of Geometric Forms* IEEE CG&A, May (1985)

31 **Faux, I. D. & Pratt, M. J.** *Computational Geometry for Design and Manufacture* First published in 1979 by Ellis Horwood Limited, Market Cross House, Cooper Street, Chichester, West Sussex, England.

32 **Fellows, J. W.** *All About Computer-aided Design and Manufacture A Guide for Executive and Managers* 1983, Sigma Technical Press, 5 Alton Road, Wilmslow, Cheshire, UK.

33 **Fishkin K. P. & Barsky, B. A.** *A Family of New Algorithms for Soft Filling*

Computer Graphics, Vol.18, No.3, July (1984)

34 **Fjallstrom, P. O.** *Smoothing of Polyhedral Models* Proc. of the 2nd Annual Symposium on Computational Geometry, Yorktown Heights, New York, Juny 2-4 (1986)

35 **Forrest, A. R.** *Lecture Notes on Computational Geometry* Computational Geometry Project, Memo CGP78/7, August (1978)

36 **Franclin W. R.** & **Akman, V.** *Building an Octree from a Set of Parallelepipeds* IEEE CG&A, October (1985)

37 **Frenkel, K. A.** *Computers, Complexity, and the Statue of Liberty Restoration* Communications of the ACM, Vol.29, No.4, April (1986)

38 **Fuchs, H., Goldfeather, J., Hultquist, J. P., Spach, S., Austin, J. D., Brooks, F. P., Eyles, J. G. Jr** & **Poulton J.** *Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancement in Pixel-Planes* Siggraph'85, San Francisco, July 22-26 (1985)

39 **Gardan Y.** & **Lucas, M.** *Interactive Graphics in CAD* First published 1983 by Hermes Publishing (France), France.

40 **Grossman, D. D.** *Opportunities for Research on Numerical Control Machining* Communications of the ACM, Vol.29, No.6, June (1986)

41 **Harris, D.** *Computer Graphics and Applications* First published 1984, Chapman and Hall Ltd., 11 New Fetter Lane, London BE4P 4EE. Published in the USA by Chapman and Hall, 733 third Avenue, New York, NY10017.

42 **Hatvany, J.** *Computer-aided manufacture* Computer-Aided Design, Vol.16, No.3, May (1984)

43 **Heckbert, P. S.** & **Hanrahan, P.** *Beam Tracing Polygonal Objects* Computer Graphics, Vol.18, No.3, July (1984)

44 **Hoffmann, C. & Hopcroft, J.** *Quadratic Blending Surfaces* TR 85-674, April (1985)

45 **Hoffmann, C. & Hopcroft, J.** *The Potential Method for Blending Surfaces and Corners* TR 85-699 September (1985)

46 **Hoffmann, C. & Hopcroft, J.** *Automatic surface generation in computer aided design* The Visual Computer (1985) 1:92-100

47 **Hopcroft, J. E.** *The Impact of Robotics on Computer Science* Communications of the ACM, Vol.29, No.6, June (1986)

48 **Hubbold, R. J.** *Computer graphics and displays* Computer-Aided Design, Vol.16, No.3, May (1984)

49 **Jared, G. E. M.** *Introduction to Solid Modelling* Fundamentals of Geometric Modelling - Review and Potential', BCS Conference Documentation, Displays Group, Cafe Royal Conference Centre, London, 26 February (1986)

50 **Kalay, Y. E.** *A relational database for non-manipulative representation of solid objects* Computer-Aided Design, vol.15, No.5, September (1983)

51 **Kalay, Y. E.** *Modelling polyhedral solids bounded by multi-curved parametric surfaces* Computer-Aided Design, Vol.15, No.3, May (1983)

52 **Karima, M., Sadhal, K. S. & McNeil, T. O.** *From Paper Drawings to Computer-Aided Design* IEEE CG&A, February (1985)

53 **Katainen, A.** *Monadic set operations* Computer-Aided Design, Vol.14, No.6, November (1982)

54 **Kjellander, J. A. P.** *Smoothing of cubic parametric splines* Computer-Aided Design, Vol.15, No.3, May (1983)

55 **Kjellander, J. A. P.** *Smoothing of cubic parametric surfaces* Computer-Aided Design, Vol.15, No.5, September (1983)

56 **Kripac, J.** *Classification of edges and its applications in determining visibility* Computer-Aided Design, Vol.15, No.5, September (1983)

57 **Kunii, T. L., Satoh, T. & Yamaguchi, K.** *Generation of Topological Boundary Representation From Octree Encoding* IEEE CG&A, March (1985)

58 **Lambourne, E. B.** *2D and 2 1/2 Models* Fundamentals of Geometric Modelling - Review and Potential, BCS Conference Documentation, Display Group, Cafe Royal Conference Centre, London, 26 February (1986)

59 **Lee, J. T. & Requicha, A. A. G.** *Algorithms for Computing the Volume and Other Internal Properties of Solids. I. Known Methods and Open Issues* Communications of the ACM, Vol.25, No.9, Sptember (1982)

60 **Liming, R. A.** *Practical Analytical Geometry with Application to Aircraft* Macmillan, New York (1944)

61 **Mantyla, M.** *Topological analysis of polygon meshes* Computer-Aided Design, Vol.15, No.4, July (1983)

62 **Mastin, C. W.** *Parameterization in grid generation* Computer-Aided Design, Vol.18, No.1, January/February (1986)

63 **Middleditch, A. E. & Sears, K. H.** *Blend Surfaces for Set Theoretic Volume Modelling Systems* Ciggraph'85, San Francisco 22-26, Vol.19, No.3 (1985)

64 **Newman, W. M. & Sproull, R. F.** *Principles of Interactive Computer Graphics* Second Edition, McGraw-Hill Book Company (1979)

65 **Noma, T. & Kunii, T. L.** *ANIMANGINE: An Engineering Animation System* IEEE

CG&A, October (1985)

66 **Ohno, Y.** *A hidden line elimination method for curved surfaces* Computer-Aided Design, Vo.15, No.4, July (1983)

67 **Owen, J.** *3D modellers: current capabilities and future trends* Fundamentals of Geometric Modelling - review and Potential, BCS Conference Documentation, Displays Group, Cafe Royal Conference Centre, London, 26 February (1986)

68 **Papaioannou, S. G.** *A Non-Orthogonal Interpolation Algorithm for NC Machine Tools* Computers in Industry, North-Holland (1985)

69 **Papaioannou, S. G.** *An application of Bertrand curves and surfaces to CADCAM* Computer-Aided Design, Vol.17, No.8, October (1985)

70 **Peckham, R. J.** *Shading evaluations with general three-dimensional models* Computer-Aided Design, Vol.17, No.7, September (1985)

71 **Perlin, K.** *An Image Synthesizer* Siggraph'85, San Francisco, July 22-26 (1985)

72 **Peterson, D. P.** *Boundary to constructive solid geometry mappings: a focus on 2D issues* Computer-Aided Design, Vol.18, No.1, January/February (1986)

73 **Phong, B. T.** *Illumination for Computer Generated Pictures* Communications of the ACM, Vol.18, No.6, June (1975)

74 **Piegl, L.** *Curve fitting algorithm for rough cutting* Computer-Aided Design, Vol.18, No.2, March (1986)

75 **Plunkett, D. J. & Bailey, M. J.** *The Vectorization of a Ray-Tracing Algorithm for Improved Execution Speed* IEEE CG&A, August (1985)

76 **Preiss, K.** *Future CAD systems* Computer-Aided Design, Vol.15, No.4, July (1983)

77 **Requicha, A. A. G. & Voelcker, H. B.** *Solid Modelling: A Historical Summary and*

Contemporary Assessment IEEE CG&A, March (1982)

78 **Requicha, A. A. G.** & **Voelcker, H. B.** *Solid Modelling: Current Status and Research Directions* IEEE CG&A, October (1983)

79 **Richards, T. H.** & **Onwubolu, G. C.** *Automatic interpretation of engineering drawings for 3D surface representation in CAD* Computer-Aided Design, Vol.18, No.3, April (1986)

80 **Riesenfeld, R. F.** *Homogeneous Coordinates and Projective Planes in Computer Graphics* IEEE CG&A, January (1981)

81 **Rogers, D. F.** & **Adams, J. A.** *Three Dimensional Transformations and Projections* Mathematical Elements for Computer Graphics (1976)

82 **Sabin, M. A.** *The Use of Potential Surfaces for Numerical Geometry* (1968)

83 **Samet, H.** & **Tamminen, M.** *Bintrees, CSG Trees, and Time* Siggraph'85, San Francisco, July 22-26, Vol.19, No.3 (1985)

84 **Satterfield, S.** & **Rogers, D.** *A Procedure for Generating Contour Lines From a B-Spline Surfaces* IEEE CG&A, April (1985)

85 **Sederberg, T. W., Anderson, D. C.** & **Goldman, R. N.** *Implicit Representation of Parametric Curves and Surfaces* Computer Vision, Graphics, and Image Processing 28, 72-84(1984)

86 **Sederberg, T. W.** & **Anderson D. C.** *Steiner Surface Patches* IEEE CG&A, May (1985)

87 **Shamos, M. I.** *Geometric Complexity* Proc. of 7th sigact Conference, Albuquerque, New Mexico, May (1975)

88 **Smith, A. R.** *Plants, Fractals, and Formal Languages* Computer Graphics, Vol.18,

No.3, July (1984)

89 **Smith, P. A.** *Solid Modelling - The Boundary Approach* Fundamentals of Geometric Modelling - Review and Potential, BCS Conference Documentation, Display Group, Cafe Royal conference Centre, London, 26 February (1986)

90 **Sutherland, I. E., Sproull, R. F. & Schumacker, T. A.** *A Characterization of Ten Hidden-Surface Algorithms* Computer Surveys, Vol.6, No.1, March 1974, by the Association for Computing Machinery.

91 **Sweeney, M. A. J. & Richard H. B.** *Ray Tracing Free-Form B-Spline Surfaces* IEEE CG&A, February (1986)

92 **Tan, S. T. & Chan, K. C.** *Generation of high order surfaces over arbitrary polyhedral meshes* Computer-Aided Design, Vol.18, No.8, October (1986)

93 **Toth, D. L.** *On Ray Tracing Parametric Surfaces* Siggraph'85, San Francisco, July 22-26 (1985)

94 **Vickers, G. W. & Bedi, S.** *The Definition and Manufacture of Compound Curvature Surfaces Using G-Surf* Computers in Industry, North-Holland (1985)

95 **Vossler, D.** *Sweep-to-CSG Conversion Using Pattern Recognition Techniques* IEEE CG&A, August (1985)

96 **Wany, K. K. & Khullar, P.** *Computer-aided Design of Injection Molds Using TIPS-1 System* Proc. of 1980 CAM-I International Spring Seminar, Denver, Colorado.

97 **Weiler, K.** *Edge-Based Data Structures for Solid Modelling in Curved-Surface Environments* IEEE CG&A, January (1985)

98 **Weiss, R. A.** *BE VISION, A Package of IBM 7090 FORTRAN Programs to Draw Orthographic Views of Combinations of Plane and Quadric Surfaces* Journal of the Association for Computing Machinery, April (1966)

99 **Wijk, J. J. V.** *SML: a solid modelling language* Computer-Aided Design, Vol.18, No.8, October (1986)

100 **Wilson, P. R.** *Euler Formulas and Geometric Modelling* IEEE CG&A, August (1985)

101 **Wilson, P. R., Faux, I. D., Ostrowski, M. C. & Pasquill, K. G.** *Interfaces for Data Transfer Between Solid Modelling Systems* IEEE CG&A, March (1985)

102 **Woo, T. C.** *A Combinatorial Analysis of Boundary Data Structure Schemata* IEEE CG&A, March (1985)

103 **Woodwark, J. R.** *The Explicit Quad Tree as a Structure for Computer Graphics* The Computer Journal, Vol.25, No.2 (1982)

104 **Woodwark, J. R.** *Solid Modelling - the Set-Theoretic Approach* Fundamentals of Geometric Modelling - Review and Potential, BCS Conference Documentation, Display Group, Cafe Royal Conference Centre, London, 26 February (1986)

105 **Woodwark, J. R.** *Blends in Geometric Modelling*

106 **Woodwark, J. R. & Quinlan, K.** *The Derivation of Graphics from Volume Models by Recursive Subdivision of the Object Space* (1980)

107 **Woodwark, J. R.** *Computing Shape* Butterworths, First published in (1986)

108 **Woodwark, J. R. & Quinlan, K. M.** *Reducing the effect of complexity on volume model evaluation* Computer-Aided Design, Vol.14, No.2, March (1982)

109 **Woodwark, J. R. & Bowyer, A.** *Better and faster pictures from solid models* Computer-Aided Engineering Journal, February (1986)

110 **Zhang, D. & Bowyer, A.** *CSG Set-Theoretic Solid Modelling and NC Machining of Blend Surfaces.* Proc. of 2nd Annual ACM Symposium on Computational Geometry, Yorktown Heights, New York, June (1986)

111 **Zhang, D.** & **Bowyer, A.** *A New Type of Blend Surface for Solid Modelling* in preparation

112 *Operator's Manual* Fanuc Systems 6M - Model B Controller

113 **Cakir, C.** *Personal Communication*

114 Cameron, S.A. *Modelling Solids in Motion*. Ph.D Thesis. University of Edinburgh, UK. (1984).

115. Parkinson. *An Automatic NC Data Generation System for the BUILD Solid Modelling System*, 16th CIRP International Seminar on Manufacturing Systems, July 1984, Tokyo.

116. Armstrong, G.T. *A Study of Automatic Generation of Non-invasive NC Machine Paths from Geometric Models*. PhD Thesis, University of Leeds. (1980). UK.